8-1-2009

Disaster recovery heuristic

Sapna Guniguntla Murthy

Disaster Recovery Heuristic

by

Sapna Guniguntla Murthy

A Thesis Submitted in Partial Fulfillment of the Requirements for the
Degree of Master of Science
in Computer Engineering

Supervised by

Dr. Roy W. Melton and Dr. Shanchieh Jay Yang
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
Aug 2009

Approved by:

Dr. Roy W. Melton
Thesis Co-Adviser, Department of Computer Engineering

Dr. Shanchieh Jay Yang
Thesis Co-Adviser, Department of Computer Engineering

Dr. Pratapa V. Reddy
Secondary Adviser, Department of Computer Engineering

Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering

Title: **Disaster Recovery Heuristic**
A Mapping Heuristic for Optimum Retrieval.

I, Sapna Guniguntla Murthy, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or part.

Sapna Guniguntla Murthy

Date

Dedication

To my family.

Acknowledgments

I would like to express my heartfelt gratitude towards the people who contributed in making this thesis possible. I would like to begin by thanking Dr. Shanchieh Jay Yang who helped me to set up a foundation for this thesis work. With his help I formulated the problem to tackle. He also helped in guiding the direction that this thesis took. Secondly I would like to thank Dr. Roy Melton who helped greatly by his unwavered interest and encouragement in the development of the thesis solution. One of the many challenges in this thesis was that it was conducted remotely. Dr. Melton fully understood the challenges that this provided and offered great support and guidance towards the fulfillment of this work. Third in the committee is Dr. Reddy who has always encouraged and helped his students to accomplish their goals. Lastly I would also like to thank Pamela Steinkirchner, a member of the office staff who helped in co-ordinating all of our efforts. This thesis has finally come to fruition with the aid and support of all these people.

Abstract

Disaster Recovery Heuristic

Sapna Guniguntla Murthy

Supervising Professor: Dr. Roy W. Melton

A need exists to develop a software simulation that demonstrates the most effective methods of evacuation for disaster scenarios. In a real-world situation this heuristic coupled with real-time data gathered by sensors could serve to provide an efficient rescue plan. Data gathered about the terrain in the immediate aftermath of the situation is invaluable in deciding a plan of action. With this type of information many different routes can be planned so that recovery or rescue can be made as optimal as possible. But of course in any rescue mission speed also is of the utmost importance. This is why we must explore heuristics that make the processing of the collected data faster. The result of this processing must be dependable and must significantly enhance the success of the rescue mission. This work proposes such a heuristic. The results obtained from this heuristic is compared with the results obtained from a process that best mimics an ad-hoc retrieval. Keeping in mind that human ingenuity can never be replaced, in this thesis we create a heuristic that will render a reliable plan of action yielding more predictable results in a disaster recovery situation. Here optimum retrieval means an act of recovery or restoration from any terrain in the most efficient way. Such a process of recovery is very useful when faced with a disaster scenario such as a hurricane or a manmade calamity on a large scale.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This research formulates a mapping heuristic that is best suited for a rescue effort. One large-scale rescue effort in recent history that comes to mind is the terrorist attacks of 9/11/2001. Let us examine this scenario in a little more detail. According to the 9/11 commission, approximately 16,400 to 18,800 civilians were in the World Trade Center complex at the time of the attacks. Within minutes of the disaster the FDNY deployed 200 units to the site, with more than 400 firefighters on the scene when the buildings collapsed. NYPD helicopters were soon at the scene, reporting on the status of the burning buildings. Doctors, nurses and other medical personnel quickly arrived at the site of the collapse to set up multiple small staging areas in the streets surrounding the site. Within hours of the incident a massive search and rescue operation was begun. The response to the disaster was almost instantaneous, but the rescue effort faced many problems in its execution. Problems with radio communication caused commanders to lose contact with many of the firefighters who went into the buildings; those firefighters were unable to hear evacuation orders. There was practically no communication with the police, who had helicopters at the scene. When the towers collapsed, hundreds were killed or trapped within. Although the brave firemen and policemen tried their very best to rescue the people trapped inside against all odds, there were about 2,600 fatalities. Not enough kudos could be given to the rescuers for their efforts, but at the same time questions arose about how any rescue process could be made faster, less ad-hoc and more reliable so that the outcome of such a situation can become more predictable and successful. Any process of rescue or retrieval involves being able to traverse all points on the terrain in the quickest and most efficient manner. A heuristic that is best able to map out such a tour can provide invaluable aid while coordinating a rescue effort. This research is a study of the viability of one such a heuristic that maps all points in the

graph so that the most efficient tour can be planned for multiple rescuers.

1.1 Background

In mathematical terms the scenario of traversing all points on the graph with least cost, is best described by the Travelling Salesman Problem (TSP). The classic TSP can be defined as follows: A salesman has to visit "n" cities with given distance d_{ij} between cities i and j, returning finally to his city of origin. Each city has to be visited only once, and the route is to be made as short as possible. A popular special case is the Euclidean TSP, where the cities are given by their positions (x_i, y_i) in the plane and the distance matrix is given by Euclidean distance:

$$d_{ij} = \sqrt[2]{(x_i - x_j)^2 + (y_i - y_j)^2}$$

The path created for a TSP is similar to what is described as a Hamiltonian cycle, but the difference between a TSP and a Hamiltonian cycle is that in TSP the tour created to visit all the cities on the map must be optimal. The optimization of the tour poses one of the most prominent problems in combinatorial optimization. It is categorized as an NP-Complete or NP-Hard problem, which means that, while a solution for a small "n" can be found fairly simply, a solution for a large "n" becomes exponentially harder to find. Before discussing the reasons for this an assumption is made that all tours between cities are symmetric to each other (i.e. any tour is the same as its reverse tour $d_{ij} = d_{ji}$). If there are two cities on the graph then only one tour is possible as shown in (refer to Figure 1.1) If there are three cities then there are two possible tours (Refer

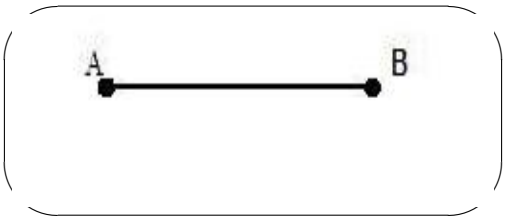

Figure 1.1: Two City Tour Map

figure 1.2). Thus for "n" cities there are (n-1)! possible tours. For the symmetric case there are $\frac{(n-1)!}{2}$ possible cases for an optimal solution. Thus for a large "n" the number of possible solutions becomes extremely large. Therefore an exhaustive search for the most optimal solution out of $\frac{(n-1)!}{2}$ solutions is impractical [3]. For this reason

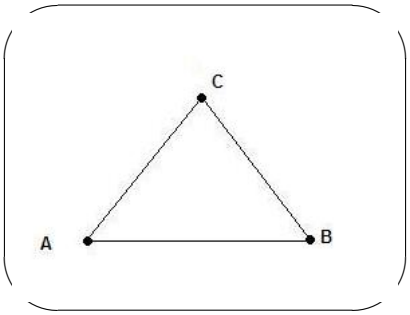

Figure 1.2: Three City Tour Map

alone instead of finding an algorithm that aims to find the shortest tour, researchers have investigated algorithms that formulate tours that are reasonably short. The two main categories of algorithms that formulate such tours are:

1. Construction algorithms: The tours are created by choosing the next node guided by some predetermined principle. Once the tours are created they are not optimized to be made any better. In this type of algorithms one finds:

   - Greedy Algorithm: In this algorithm a tour is created by choosing the closest node to the node being "visited" currently. It is also called the "Branch and Bound" algorithm [10] . According to the results gathered by Robert Dakin, one of the many researchers in this field, the greedy algorithm is quite fast up to 10 nodes but becomes painfully slow beyond 40 nodes.
   - Insertion heuristic: This algorithm starts with a subtour where a tour is created between a few cities on the map. Then new nodes are inserted into the subtour such that the cost of the subtour is minimized. The process of insertion is stopped when all cities on the map have been visited.

2. Optimization algorithms: These types of algorithms take a basic completed tour and optimize it using various procedures for example choosing different combinations of insertions so that the cheapest tour is found. Optimization algorithms are fast for problems up to 100 nodes[10].

1.2 Objectives

The disaster recovery heuristic will try to come up with an insertion type algorithm that will create a reasonably short tour so that a rescuer will be able to "visit" the victims in

any disaster struck area in the shortest time/distance. Expanding on this basic premise, this paper will attempt to demonstrate how different variations of the classic TSP show even bigger correlation to the outlined problem. Studying these variations will give a better insight to the mathematical challenges of the Disaster-Recovery algorithm. There are several interesting variations of the classic TSP. Two popular ones that show relevance to our problem are:

1. PTSP: This is a variation of the classic TSP involves one salesman and multiple tours. The cities in the graph have to be visited not once but m number of times. The visit days and tours have to be planned for each day such that the total distance traveled is minimum. PTSP, an NP-hard optimization problem includes both temporal and spatial decisions at the same time [10]. PTSP is pertinent because it is possible that one rescuer has to conduct several tours to make a complete recovery. Planning m tours such that each rescuer can cover his terrain in the shortest time and distance can be one method of planning a cohesive rescue effort.

2. VRP: This variation of TSP involves m salesmen and is therefore also known as mTSP. It consists of finding m vehicle routes of least total cost. It does not involve temporal planning like PTSP. VRP is a well known integer programming problem which falls into the category of NP Hard problems.The VRP arises naturally as a central problem in the fields of transportation, distribution and logistics [4]. Vehicle routing TSP is pertinent to the defined problem because with this algorithm a group of rescuers can be coordinated to traverse a map in the most efficient manner possible.

In conclusion it can be seen that creating a solution for a small rescue effort (less than 50 people, or 50 points of 'visits') may be relatively simple to calculate. However for large-scale disasters like earthquakes and floods, large recoveries have to be coordinated in the shortest possible time. Here creating a solution becomes a significant mathematical and programming problem. In addition there could be several variations and possibilities for each disaster scenario. This study amalgamates some of the previous efforts in the field of TSP and transposes their ideas for a new hybrid solution that achieves with reasonable success, all the proposed goals for a disaster recovery.

In the next chapter related works on TSP and other studies for gathering data from the field are discussed in detail. The studies that have been picked to be examined in detail help ultimately in formulating the problem statement and the solution for it.

Chapter 2

Summary of related work

Since TSP is such a classic problem, there are several papers that propose various algorithms that give a result close to the optimal solution. The papers explored during the course of this research are the ones that have unique approaches to the problem and will be a guide to finding a solution to the thesis problem. The ideas in these papers are discussed briefly while explaining how solutions can be formulated and improved on the researcher's ideas to obtain a projected solution.

2.1 Classic TSP - ENLS approach

In his paper "Exponential neighborhood local search for the TSP" Gutin [7], proposed that a local search mechanism of formulating a tour is one of the most practical ways to find optimal solutions on a large scale. The paper explored 'Exponential neighborhood local search (ENLS)' which finds the very best local tour in a large exponential number of tours. While exploring such a solution he discusses a new way to evaluate heuristics proposed by other researchers in the field, Glover and Punnen [7]. They proposed something called a domination number:
domn($\mathcal{A}$, n) = d. where $\mathcal{A}$ represents the heuristic, n is the number of cities on the digraph. 'd' is the integral value which is a function of n [d=d(n)] such that for every instance of TSP on the graph i.e. every tour generated by the heuristic $\mathcal{A}$ is not worse than at least d tours. It thus follows that higher the value of d which is domn($\mathcal{A}$, n), the better the algorithm. To elaborate, a heuristic with a larger domination number is a better one as compared to a heuristic with a lower domination number.The greedy

algorithm has a domination number that is significantly lower than (n-2)! With this new method for calibration Gutin goes ahead and proposes ENLS. The main idea behind this algorithm can be simply explained as follows.

A weighted graph G with n vertices is considered. A Hamiltonian cycle of minimum weight is graphed out in the plane. In a regular local search, the plane is divided into neighborhoods of small polynomial size to find a near-optimal solution. However the new approach tries to find solutions by finding the best amongst a large set of tours in polynomial time.

Its approach is to divide the graph into two parts: x coordinates and y coordinates. One set of coordinates $\{y_1, y_2, y_3...\}$ has more points in it's set than the other set of coordinates $\{x_1, x_2...\}$ (Refer figure: 2.1) First a subtour is created with y coordinates

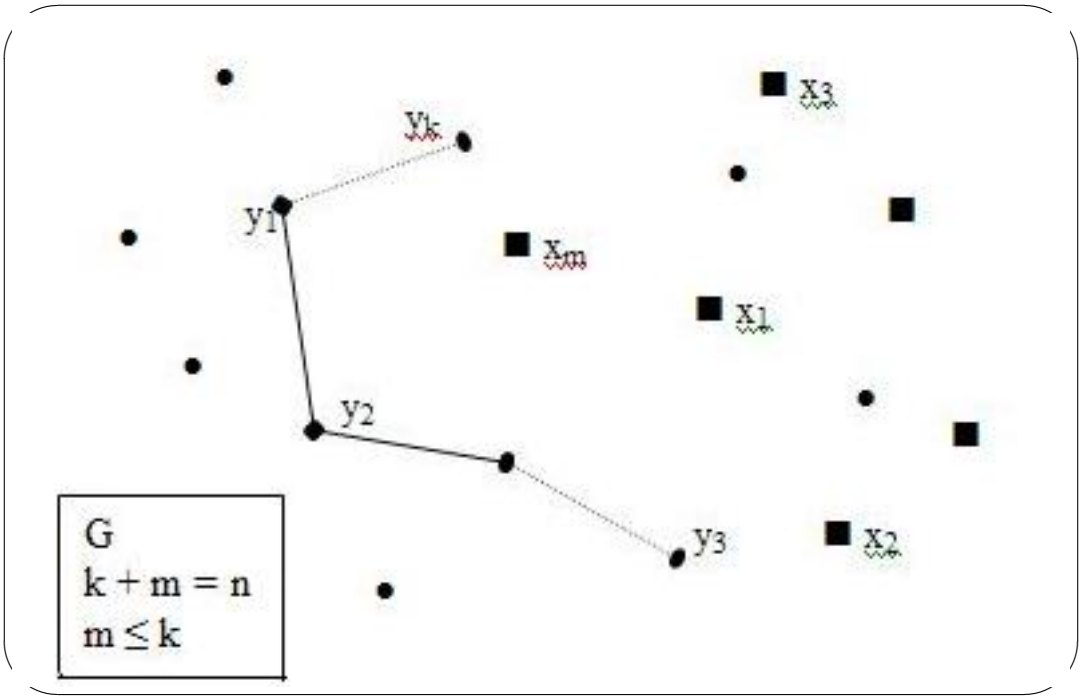

Figure 2.1: Bi-partite Graph

and later x coordinates are inserted into the tour such that there is one x between every two y. All possible tours created this way are collectively called a neighborhood N and symbolized by:

$$N = N(y_1, y_2, y_3.....y_k; x_1, x_2, ...x_m).$$

Using an assignment algorithm one can find the best (of least weight) among these tours in the neighborhood N in O(n^3) time. A set of fictitious cities in the above set of x cities

is created so that the process of inserting cities in the y tour is simplified. Their insertion is accounted for by simply disregarding their distances. Now a bigraph is created whose weight is calculated such that if there is a 'real' x then it is inserted between y_j and y_{j+1} and the distance is calculated. If the x is fictitious then $d(y_j, x_{\tau(j)}) = 0$. So it follows that weight of tour in G equals the weight of matching tour in B (the new constructed tour after insertion). Thus to find an optimal tour, it suffices to construct a minimum weight perfect match in B, the complexity of which is O(k^3) = O(n^3). Gutin further proves that for a particular value of m for the set x, the number of tours obtained in graph G is maximized. Now this is compared to a greedy algorithm with a complexity of roughly O(n^2), the ENLS approach may seem to consume more resources (time, CPU cycles etc). However this approach cannot be blatantly compared to the linear approach of a greedy algorithm since ENLS breaks down any large neighborhood into a series of blocks. The assignment algorithm of order O(n^3) finds the best path in that block/neighborhood. Since the permutations for the order of insertions are performed for a far smaller number of cities as compared to the linear approach, Gutin proposes that ENLS is a more efficient algorithm to find the best tour.

From the view point of the disaster heuristic the most important idea presented in the paper is that areas can be localized and transpose the graph such that fictitious branches can be inserted. These branches will help mathematically deduce the lowest-weight perfect match. Being able to permute combinations in this localized manner limits the exponential curve of processing for a large area or number of cities. Also the idea presented in this paper is very close to any real-life scenario where rescuers have to divide and conquer in order to carry out rescues. As a side note it can be seen that the permutations being carried out in the ENLS approach lend themselves very well to parallel computing [7]. Parallel processing is a good way to reduce processing time to achieve required results.

2.2 PTSP - How it relates to the problem

The previous algorithm is a close approximation of a real-life scenario where a disaster zone may be divided into smaller more manageable regions where local tours can be generated. Constraints of distance are not the only variables in a TSP. There are also constraints of time where many rescuers have to be coordinated to visit a location a certain number of times. In the domain of TSP, problems with constraints of time are classified as "PTSP" problems. In a PTSP a traveling salesman has to visit each city

i, r_i number of times over a given planning period m. For a solution a given set of combination of days is formulated that the salesman can visit the city. Thus the problem includes making temporal and spatial decisions such that the total distance traveled over the planning period m is minimized.

Bertazzi, Paletta and Speranza proposed a paper called "An improved heuristic for the PTSP". Like their peers who proposed an initial solution with an improvement procedure, these authors have also presented a construction type algorithm with an embedded improvement procedure.

The algorithm is explained as follows. The goal of the algorithm is for a traveling salesman to visit each city i of a given set of cities, $I = \{1, 2, 3...n\}, r_i$ times over a given period m. After the planning is done a set of combination of days for each city is determined $V(i) = \{C_1, C_2, C_3...C_{ni}\}$ where each C_k specifies a particular combination of days. Thus each C_k is made up of r_i days.

The given parameters of this problem are: the salesman starts from city(0), makes his tour, and returns to city(0). Thus every tour starts and ends with $V(0)$. At least one other city needs to be visited in every tour for the tour to be considered valid. The distance of every arc between cities (i and j for example) is symmetric (i.e. $d(i, j) = d(j, i)$). The algorithm first goes through what is called the 'City Processing Procedure' which assigns to a not yet processed city a combination of visit days $V(i) = \{C_k\}$. The combination of days may be Sun, Tues, and Wed for example where $r_i = 3$. Now for each day of this combination, a city is inserted into the corresponding tour. This move is much like the insertion mechanism used in the sub-tours discussed in [7], the only difference here is that the insertion starts with the combination days instead of the sub-tour (temporal vs. spatial). The 'City Processing Procedure' is iterative until all cities have been assigned a combination of days.

This iterative process is temporarily interrupted every time the number of processed cities is a multiple of a given parameter unique to this algorithm which is $\frac{n}{p_1}$ where p_1 is the number of times the procedure has been interrupted and n is the total number of cities. In the first iteration the process will be interrupted when all cities have been processed. When interrupted a second process called the 'improvement procedure' is run on the cities that have been processed so far. The solution that has been formed thus far, by assigning a combination of days and then inserting a city in every day's tour, is improved by removing a given number p_2 of processed cities and assigning to them a new combination of visit days.

After this reinsertion process is finished the 'City Processing Procedure' continues to iterate through the cities until the interrupt condition occurs again. This entire phase is called the tour construction phase.

The validity rule is that at least one city other than $V(0)$ needs to be visited for an acceptable tour. If during the tour construction at least one tour is obtained which is not valid, then a process called the 'Feasibility Procedure' is applied to the obtained solution. In this process one city which belongs to a tour with at least two more cities in it is selected. This city is then removed from the tour that includes it and a combination of visit days are assigned to it such that the combination includes the largest number of empty tours. This process is repeated until there are no empty tours remaining. Thus the process converts an infeasible solution to a feasible one.

Finally a process called 'Modified Improvement Procedure' is applied to the solution to improve the quality of the result. The modified improvement procedure is similar to the improvement procedure and is applied to tours with at least two cities in the tour. This clause preserves the feasibility of the solution. The results obtained by this algorithm is shown to be not worse than the best known result in terms of total distance in 95% of the instances and is shown to be strictly better in 45% of the results [8].

In this scenario on different tour days, different tour cities need to be visited r_i times. To get this result a plan is created to achieve the shortest tour in the least amount of time. This problem is very relatable because in particular rescue cases it could be important to calculate a tour that accounts for repeated visits to a location in the least amount of time.

2.3 Vehicle Routing Problem - Tabu and GENIUS Algorithms

The next realm of TSP is called the mTSP or Multiple Traveling Salesmen problem. This set of problems is also called vehicle routing problems as these find direct applications in the field of transportation and logistics. Here multiple salesmen/vehicles need to be dealt with, each of which is given a route. The objective is to determine m routes of least total cost. However, unlike PTSP we do not need to worry about temporal planning for these m routes.

Franca, Gendreau, Laporte and Muller authored a paper is titled "The m-Traveling Salesman problem with Minmax Objective" that deals with mTSP[8]. The minmax objective refers to the goal of minimizing the length of the longest route and thereby achieving the objective of a 'mTSP'. Let us start by defining the problem. Consider a

graph $G = (V, A)$ where $V = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ is the set of arcs in the graph. Each one is associated with a cost (distance) which is represented by a matrix of non-negative values. The distances are bound by the laws of triangle inequality i.e. $C_{ij} \neq C_{ik} + C_{jk}$ and are considered symmetric only if $C_{ij} = C_{ji}$ for all j and i in the matrix. V_0 is the depot where all tours originate. Tours have to be formed such that every city $V_i \in \frac{V}{V_0}$ is visited exactly once.There are possibly two ways to look at the problem:

1. An mTSP can be converted into 1-TSP (i.e. a regular TSP problem). The solution can then be divided into m-routes. One method of doing this is to introduce false nodes to simulate the depot so that disintegration is possible.

2. The other methods is to deal with mTSP as is. Create m routes simultaneously by least cost insertion or nearest neighbor criterion and look to reduce the cost of mTSP.

Computational results obtained by Laporte and Nobert have indicated that the second method is superior for finding a near-optimal solution [8]. A popular search heuristic is called the 'Tabu' heuristic. This heuristic repeatedly takes a solution and moves it to best solution found amongst its neighbors. This movement is possible by ignoring or relaxing the objective to allow exploring the solution space. Thus new solutions are formed. However in order to prevent the same neighborhood solution to be encroached on time and time again, once a solution is examined it is moved into a Tabu list. The solutions contained in this list may not be touched by its neighbor. Thus the solution space is narrowed down making it less time consuming. The Tabu search method is very customizable and can be tailored to fit most TSP problems. In this paper, Tabu search algorithm embedded with GENI (stands for GENeralized Insertion) insertion method is used to arrive at a near optimal solution for the minmax objective.

The following is a description of the generic GENIUS algorithm. First an initial tour solution is created. Neighbor solutions are then explored by repeatedly removing a node from the current tour and inserting it into a tour containing the closest neighbor by means of GENI.In a separate study GENI is found to be more powerful than standard procedures, since insertion is only done on tours containing its closest neighbors and while inserting, that tour is also re-optimized for shortest tour. The part where the GENI insertion becomes less myopic than other insertion methods is that in the process of creating a tour, not only the shortest tour is created between the two types of insertion but that the best possible candidate for insertion are computed amongst all the neighbor vertices. The complexity of this computation is $O(np^4 + n^2)$. The insertion method is

then followed by a 'US'(stands for Unstringing and Stringing) post-optimization. This process involves removing each vertex in turn from a formed tour by running a reverse GENI and then reinserting using the GENI algorithm. Those two procedures together are called the 'GENIUS' algorithm.

In order to apply this to the specific problem at hand, an adaptation requires the concepts of 'Global neighbors' and 'Local neighbors'.

Global neighbors - These vertices are defined as the q closest predecessors and q closest successors of v among all the vertices $\frac{V}{\{u\}}$, where q is an input parameter. Each vertex v also has $2m$ local neighborhoods, two for each route.

Local neighbors - These vertices are defined as the set of p closest predecessors and p closest successors of vertex v on a path k. Here p is an input parameter. The sets of local neighbors of a given vertex are updated dynamically as the algorithm progresses. (refer to figure:2.2)
Global neighbors of $v = \{v_4, v_3, v_2, v_1, v_6\}$
Local neighbors of v with route 1 = $\{v_4, v_3, v_2\}$
Local neighbors of v with route 2 = $\{v_6, v_1, v_8\}$

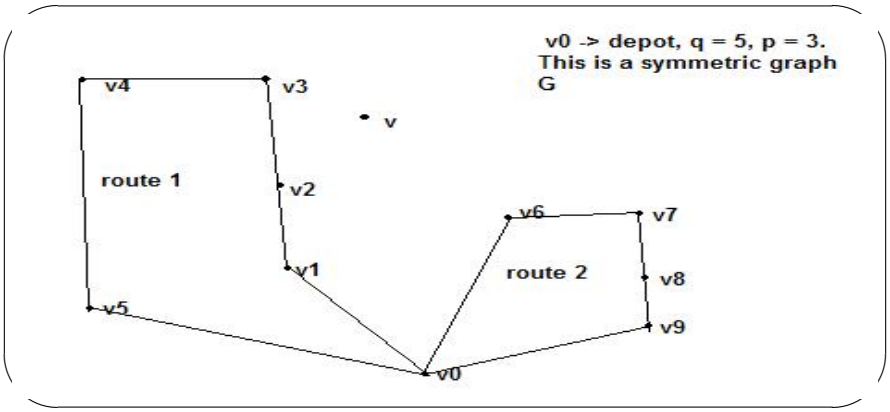

Figure 2.2: Local vs Global Neighbors (Original non-optimized tours created by greedy algorithm.)

The adapted heuristic is as follows:

1. Obtain global predecessors and global successors for each vertex v_i.
   In a symmetric graph for $q = 5$(number of vertices in tour), the predecessor and successor set is $\{v_{i+1}, v_{i+2}, v_{i+3}, v_{i+4}, v_{i+5}\}$

2. Randomly select m vertices and construct a return route between each vertex and

depot. Thus at the onset m routes are created. For each route m obtain local neighbors for each vertex v_i.

3. Now for every v_i and a tour k use GENI to form a route that has the least increase to route size. At this point a new tour has been formed. Update local neighborhood lists for all vertices that are uncharted in a tour.

These steps for an initial solution that forms m tours with no remaining uncharted vertices in the graph. Now the solution is subjected to optimization to achieve the minmax objective of shortening the longest tour.

1. Iteration t=0 and the longest tour size of the m tours is recorded as $\overline{z}.z* = \overline{z}$. At this point no move is in the Tabu list.

2. Identify the longest tour as k. For each vertex on the route v_i, identify a set $R(v_i)$ of different possible routes containing v_i that are not k and not on the Tabu list. The constraint also for these new tours is that it has to contain one global successor and one global predecessor of v_i. For each route in the set of $R(v_i) = \{(v_1, v_2, v_i, v_{i+1}, v_0), (v_2, v_1, v_i, v_{i+1}, v_0), (...), (...)\}$ determine Z_{il} which is the length of the longest route.

3. If $S(k)$ is the set of vertices of route k, then identify V_i* as a vertex of the set and $l*$ as a route in the set $R(V_i*)$. If Z_{i*l*} is the minimum size route of all $V_i \in (S(k))$ i.e. $Min_{l \in R(Vi)}\{Z_{il}\}$. If a good tour has been made, move V_{i*} from route k to route $l*$ using GENI insertion. Now that $\overline{z}$ no longer applies, update $\overline{z}$ with current longest tour size. If now the updated $\overline{z}$ is less than its old value $z*$ then update $z* = \overline{z}$(the new low value). If $Z_{i*l*} < \overline{z}$ get new value of $\overline{z} = Z_{i*l*}$. Declare move of V_{i*} from $l*$ to k as Tabu until iteration $t + 0$ since clearly the shorter route is created with V_{i*} in $l*$. Update lists of successors and predecessors with view of new route and move to next iteration $t = t + 1$ and repeat.

After this re-adjustment of tours with minmax objective the algorithm proceeds to the post-optimization procedure. This process involves the following steps:

1. This step identifies the best known solution thus far arrived at and takes the longest tour k from m tours.

2. Apply the 'US' procedure as described earlier to arrive at a solution. Identify longest route in new solution and call it l. If $l = k$ then stop, else set the longest route to now found route $k = l$ and repeat.

The authors comment that in order to produce quality results the choice of p (global neighbors) and q (local neighbors) must be made properly. Here $p \propto \sqrt{n}$ but never less than 3 and similarly $q = [\alpha\sqrt{n}]$ where $\alpha = 0.6$ (proved empirically). Lastly the 'US' post-optimization shows significant improvement in cases where the longest routes consist of a large number of vertices [11].

What is obtained from this paper are the ideas of using a minmax objective and a Tabu list for shrinking the solution space. The 'GENIUS' algorithm widely used in vehicle routing problems is a well established and proven technique that can be utilized in the thesis problem to provide good quality results.

2.4 MACS-VRPTW

The formation of multiple routes is not the only concern when the disaster heuristic is examined. It is a step in the right direction but what it is also necessary to introduce the constraint of time. An mTSP or a vehicle routing problem with this constraint is formally called VRPTW.

Besides the 'GENIUS' algorithm and straight-cut 1-TSP divided into m-routes, another interesting approach to vehicle routing problems is known as the 'Ant Colony Optimization' or ACO. This new class of natural algorithms is inspired by the foraging behaviors of natural ant colonies. A path making heuristic relates to the foraging habits of ants is that we are attempting to create paths in the real world, and while creating paths empirical results have shown that topology is a critical factor. The ant colony methods work through a system of cooperation and gather useful topological information that is used to update nodes and decision making processes in real-time.

A paper called "MACS_VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with time windows" authored by Lugano deals with one such algorithm.

The MACS_VRPTW is organized with a hierarchy of artificial ant colonies designed to optimize a multi-objective function. This means that this is a system to resolve two objectives based on priorities. In this paper one objective is to minimize the number of vehicles or number of tours. The second objective is to minimize the distances traveled. Besides this system of hierarchy another system that is introduced is cooperation between the two systems or colonies by the exchange of information. Amongst real ants this exchange of information is conducted by depositing pheromones on their food trails. This acts as a collective memory for the ants that are currently foraging, thus

making this process more efficient. This type of low-level communication is an effective method of creating good path solutions. This translates into the world of computing in the following manner: if ants were parallel processes slaves to a master process which acts as the collective memory. In the parallel processing world, the slave units have to compute path solutions while the master unit executes and controls the final path solution being built.

VRPTW seeks to minimize tours, the other objective is to minimize total time (time window).The definition of this problem is as follows: n cities have to be serviced from one depot, each city has a certain quantity of goods $V_{i(i=1,2,...n)}$, and each vehicle has a capacity of Q. This algorithm introduces the concept of capacity, and thus this form of VRP is also called capacitated VRP or CVRP. Since each vehicle is limited by its capacity Q, it has to return to the depot to reload. Each city must be visited only once and can only have a demand of maximum Q. If G is the graph with vertices set $C = \{c_0, c_1, c_2, ...c_n\}$and arc set $L = \{c_i, c_j\} : c_i, c_j \in C, i \neq j$. Let c_0 be the depot. Each city c_i has an associated quantity q_i associated with it. With each arc(c_i, c_j) there is an associated t_{ij} which is the time taken to travel between the two nodes.

The goal is to find a set of tours of total minimum time. To adapt the CVRP to the VRPTW a time window is added to the list of constraints $[b_i, e_i]$ for each customer c_i. The customer has to be served between beginning time b_i and end time e_i. Thus the capacity of the truck is fixed, and the number of vehicles v is not. It is one of the common objectives to minimize v.

To solve these problems (limit is upto 100 customers/cities) there are a number of exact methods which are currently used by the transportation industry. The heuristic methods are traditionally used for a larger customer base and work best in an adaptive customer scenario.

ACS is explained as follows:

TSP is a problem where a shortest tour amongst all cities has to be found. ACS is applied to this problem by associating with each arc values (n_{ij}, t_{ij}) where n_{ij} is the closeness between the two nodes and t_{ij} is the desirability of that arc. $N_{ij} = \dfrac{1}{d_{ij}}$ is a static value while t_{ij} is a dynamic value that is updated through the course of the algorithm. The stronger the value of t_{ij}, the better is the probability for this arc to be chosen. The ACS goal is to find the shortest tour m. Ants build tours in parallel. Each ant is randomly assigned to a node, and it has to start building a tour. Each ant builds a tour by adding nodes to its existing tour. The selection of the next node to be added is done probabilistically in the set N_{ik}, the set of remaining uncharted nodes for the current tour.

The probabilistic rules include the probability of exploitation and the probability of exploration. Both probabilities are calculated, and the heuristic evaluation of the results is obtained, which is q_0 (the probability). This value is the relative importance of exploitation vs. exploration (p_{ij}) Either of these probabilistic rules can be used depending on the value of q_0. Smaller the probability of exploitation or usage for the node, then more the rule p_{ij} is used to evaluate a node's candidacy for the tour.

Each ant adds nodes to its tour one by one in this manner until all nodes are included in the tour. Once all ants have finished tour construction, solutions are tentatively improved using local search procedure. Then the process is to update the pheromone or desirability list by identifying the best tours. The process repeats with the new desirability list until termination conditions occur. The termination condition could be that a fixed number of solutions have been generated, a certain computation time limit has been reached,or in the best case no improvement has been achieved over a set number of last few iterations. Thus paths/tours have been created that satisfy the minmax objective. Thus we have a good solution for TSP.

ACS uses both local and global updates to the pheromone data. Locally when an ant uses an edge/arc it reduces its t_{ij}, thus other ants are less likely to use it. This decreased set of possible paths encourages exploration, other routes are now explored to be added into the other tours. Globally the best solution is used to increase t_{ij} to show that the shortest routes between two nodes have been found. This is how ACS creates optimized tours for TSP. Let us apply this to MACS-VRPTW with two objectives:

1. Minimize number of tours .

2. Minimize total travel time.

Goal 1 takes precedence over goal 2.

To achieve defined dual objective we need two ACS colonies. The first one ACS-VE1 will lessen the number of tours, and the second ACS_TIME will shorten the time. Both have their own pheromone lists but share the same ψ^{gb} (best solution tour) which is managed by MACS-VRPTW.The initial solution ψ^{gb} is the tour constructed neighborhood greedy algorithm. ACS_VE1 takes a solution and seeks to improve it by reducing the number of vehicles used by one. ACS_TIME then takes over the modified solution and tries to improve the time used in the existing number of routes. Thus ψ^{gb} is updated and iterated through for improvements.

In summary both ACS-TIME and ACS_VE1 start at a random copy of the depot and start constructing tours that do not violate the constraints of the tour which includes constraint (b_i, e_i). In each step a new node is added to the tour using the probabilistic

laws. If closeness n_{ij} is considered then not only is the distance between the two nodes dij included but also the time window $[b_i, e_i]$ associated with node j. For ACS_VE1, the number of times j has not been included in a tour (In_j) is also a factor considered before choosing the next node. At the end of this process an unfeasible solution may be formed. This partial tour is completed using insertion while looking for shortest tours. This method is competitive with the best existing methods for producing quality results in relatively short computation times [9].

For the thesis problem ideas of executing multiple objectives and updating current desirability information at run time is obtained. Another important factor that this paper presents is the idea of multiple depots. This setting is much closer to a real life scenario where there are multiple enter and exit points, especially in a disaster zone.

2.5 Research - TSP results

In the paper 'Comparison of TSP Algorithms' [1], the authors Byung-In Kim, Jae-Ik Shim and Min Zhang, have implemented variuos well know TSP algorithms and studied the results they produced from these heuristics. The well know TSP algorithms that they implemented in this work are:

1. Greedy Algorithm - This is the simplest algorithms in the category of 'improvement algorithms'. A tour is created by attaching the next closest node. Thus at the end, a short tour is created.

2. 2-Opt Algorithm - Initial tour solution is created. It is then improved by using a 2-opt move. In a 2-opt move, two edges from the solution are removed and then reconnected to form a valid tour.

3. 3-Opt Algorithm - The 3-Opt algorithm works in a similar fashion to the 2-Opt, in this after the initial tour solution is created, 3 branches are removed and reconnected to improve the solution. It can be defined as two or three consecutive 2-opt moves. Thus 2-opt algorithm is a subset of the 3-opt algorithm.

4. Simulated Annealing - It is defined as a randomized search algorithm in which moves with negative weights are allowed.

5. Genetic Algorithm - This type of algorithm is inspired by the natural evolutionary process. The rule survival of the fitest is applicable here. Every path has a fitness

measure associated with it. Depending on the fitness of the path, it is selected to create a tour solution.

6. Neural Network - These are also know as 'Self Organizing Feature Maps'. One vertex in the map is selected to begin the tour. The tour then is grown by a node selection process. There are multiple iterations and at the end of that a good solution is created.

The results generated from all these algorithms are compared with the results generated from the thesis heuritics.

2.6 Other related fields of study

Societal disasters and recovery of people and materials after a large scale disaster comes under the category called emergency management. The first phase of emergency management is called 'Search and Rescue'. It is in this phase that my thesis will attempt to be most useful.

Other research that concentrate on this area of study:

1. Adaptive Clustering for Mobile Wireless Networks. Authors: Chunhung Richard Lin and Mario Gerla.[2]

   This paper talks about radio packet multihopping which is a fast deployable wireless infrastructure used for several applications including disaster recovery search and rescue. It proposes a multicluster algorithm which enables the different nodes in the network to 'talk' to each other to determine resources available and its required allocation.

2. Scalable Routing strategies for ad-hoc wireless networks. Authors: Atsushi Iwata, Ching-Chuan Chiang, Guangyu Pei, Mario Gerla, Tsu-wei Chen.[6]

   This paper talks about a large number of mobile stations that are interconnected by a multi-hop network; they propose 'Fisheye state routing' and 'Hierarchical state routing' schemes which maintain the integrity of the network under mobile conditions.

There have been several proposals for multi-hop network management schemes which enable better and more reliable transfer of data back to the clusterhead or the

master node that manages all this data. The scheme implemented in this paper is different in the sense that it maps out an appropriate route after this data has been gathered. While gathering the data is step one of the rescue process, the manipulation of this data to deliver a reliable plan of action for rescue is step two. The disaster recovery heuristic is this step two.

In the field of applying TSP with constraints, here is another paper: Resource-Constrained Geometric Network Optimization (1998). Authors: Esther Arkin, Joseph S. B. Mitchell, Giri Narasimhan.[5]

The authors study a graph with several network optimization problems, given a certain resource bound B. For example they study a regular TSP where the salesman has to visit points on the graph in the most optimal way given that he has a resource limit. He has to construct a tour such that his resource (say fuel) doesn't run out and he has to cover all the points on the graph in the shortest time/distance.

This paper is similar in its concept to the defined problem in that there are real-world constraints of personnel available, different entry and exit points, time limitations etc. This thesis is unique in that it applies some of these real-world constraints to find an optimal tour by using TSP. This concludes the research portion of the paper where many different ideas and approaches to the defined thesis problem have been explored. In the next chapter we will discuss the Disaster Recovery problem statement, and how the research plays into formulating a solution for it.

Chapter 3

Disaster Recovery Heuristic - Problem definition

Problem Statement

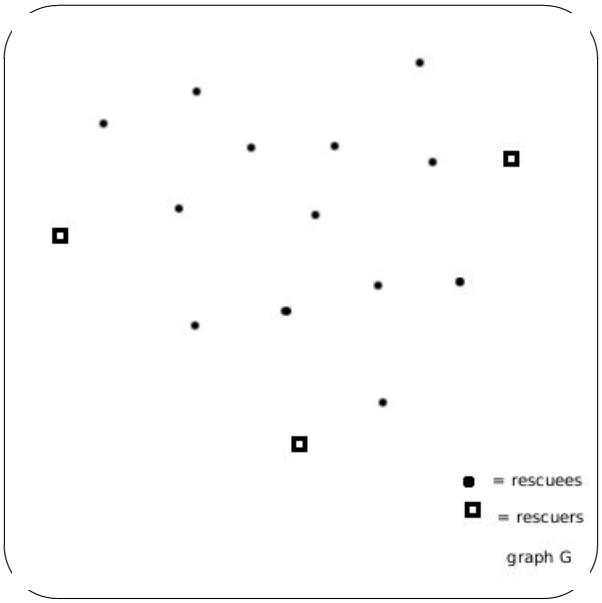

Figure 3.1: Rescuers and Rescuees

Given:

1. Assume that there are m rescuers and n rescuees (refer figure:3.1).
2. Positions of all m rescuers and all n rescuees are known initially.
3. Shortest path between any two rescuees is defined by arc-set C_{ij}.

OBJECTIVE

This problem has a minmax objective. Minimize the maximum distance and time taken by any rescuer to rescue his sub-group of rescuees such that in effect the total time/distance taken by the m rescuers to rescue all n rescuees is minimized.

In order to find a solution to this problem one can approach this as an mTSP problem or a PTSP for m rescuers. The solution includes creating different localities in the graph and then creating the shortest route for each locality. This has been achieved by using a well-proven heuristic called GENI. This heuristic has been discussed in detail in the earlier chapter"Summary of related work", in section"Vehicle Routing Problem - Tabu and GENIUS Algorithms". Creating the shortest possible local tours in turn helps create a reasonably short total tour. This solution however can be further optimized. Here is a brief description of the optimization process. The largest local tour is identified along with the local tour with the shortest distance. Then the vertices (cities) with the longest distance is identified and conversely the cities with the shortest distance is identifies in the short local tour. The vertices(cities) are then exchanged. They are extracted from their local tours and inserted in the opposite tour by means on the GENI algorithm. The tours are regenerated with the new vertices and the total tour is once again calculated. This process of optimization is repeated until we find a tour that is shorter than the originally calculated total tour or until we reach a certain pre-determined number of trials. To compare the results of this heuristic, a Greedy algorithm and a simple Local Search Heuristic is also implemented.

The greedy algorithm has been implemented by taking one rescuer and creating a tour. The tour is created by adding the closest next vertex to the tour. The closest vertex is determined by an extensive search and calculating the shortest distance from the current vertex on the tour. Thus the total tour created is a reasonably short tour. This works out very well for a small number of rescuers and rescuees. However the time taken for large number of rescuers and rescuees makes it an unrealistic option. However for the purposes of comparing results it shows how far away the distance calculations are from the resukts as generated by the greedy algorithm. It will also serve to show how much of an improvement the time taken for calculations are when comparing the DRH results from the greedy algorithm results.

The Local Search Heuristic has been implemented by taking m rescuers and dividing up the graph into m local areas. Each rescuer creates a tour by visting all the

cities/rescuees in his assigned local area and then coming back to the depot. The local tours are created by the greedy algorithm in each local area. The total tour is the sum of all local tours. This heuristic was implemented because it closely approximates the scenario that we are considering the DRH. Thus it gives us a good base for comparison which is between the result created with the greedy algorithm with one rescuer and the DRH. Eventually when the number of rescuers and rescuees is very large, this algorithm too will fail in the time taken for computation because it inherits all the drawback of the Greedy Algorithm.

In addition to the static nature of the problem, one must acknowledge that in any real-world situation real-time data plays an important role in the decisions taken by the rescuers. This factor will be discussed in the MACS-VRPTW algorithm. Adapted to the case at hand, one can see that as the rescuers finish their local tours they can update the desirability of their routes to make the tours easier for other rescuers. The latter part of optimizing during real-time is an extension of this algorithm and will not be explored here.

3.1 Description of Disaster Recovery Algorithm

A detailed flowchart of the DRH is attached in the appendix for reference.

The algorithm starts with asking the user which algorithm should be used to process the data. The choices are:

1. Greedy Algorithm
2. Local search Heuristic
3. Disaster Recovery Heuristic

It then asks the user for the number of rescuers and rescuees. Since the thesis algorithm is aimed at large numbers of rescuers and rescuees, it creates a map by assigning randomly assigning distances between the co-ordinates or rescuees. In a realworld scenario this would be approximated to the locations where the rescuees have been trapped etc. In this scenario the rescuers will have to traverse through these points in order for a rescue to take place. This data is common for whichever algorithm the user has chosen to process the data to create tours.

3.1.1 Greedy Algorithm

This is a very simple algorithm where there is only one rescuer to traverse the path in the graph. This has been implemented to show the most basic TSP algorithm and also because it provides as its result a reasonably short path through the map. In computing this we obtain a baseline to compare (in terms of distance) how good the Disaster Recovery Heuristic results are. The drawback of this method is that it is the costliest in terms of time taken for calculation, and is therefore impractical for use in the real world scenario.

There are two methods for this algorithm. The first one is very simply implemented by taking into account the distances between all points(rescuees) and creating all possible tours. The shortest tour is then picked. The second method is that we create the tour by considering the next closest vertex. This vertex is chosen after exhaustively determining which point on the graph is closest to the current vertex on the tour. Thus at the end, the shortest possible tour is constructed. For this thesis this second method has been implemented.

Because of the nature of this algorithm it is also called an 'Exhaustive Search Method". The following chapter which displays results will show that it creates a comparitively good result in terms of distance for any number of rescuees. However because of its exhaustive search principle, the time taken to obtain results increases exponentially as the number of rescuees increase.

3.1.2 Local Search Heuristic

This algorithm has been implemented to bring the real world scenario closer to the mathematical world. In this we consider multiple number of rescuers as well as an increasing number of rescuees. In this manner it is also closer to how the problem statement of Disaster Recovery Heuristic has been formulated. But in this algorithm the real world scenario is also mimicked in the sense that each rescuer sets out to rescue a random group of rescuees. In this manner there are as many groups of rescuees as there are rescuers. If the number isn't evenly divided we then assign the numbers as evenly as possibly between the rescuers. These rescuers now set out to traverse the randomly assigned rescuees in the group that they are given. For the purposes of the algorithm each group is called a locality - the locality being defined as being made up of the rescuees co-ordinates on the map. Each rescuer then creates a tour within his locality by using the greedy algorithm. The rescuer sets out to construct his tour by identifying the next closest vertex within his locality and thereby creating a reasonably short tour for

his locality. The total tour for the whole graph is thus theoretically a reasonably short tour calculated by summing up all the local tours for all rescuers.

This algorithm is better than the Greedy algorithm because it divides up the exhaustive search between 'n' rescuers. But in doing so it also inherits the defects of an Exhaustive Search Algorithm. It works well for a small number of rescuers and rescuees but then the time taken for calculation becomes exponentially large as these numbers increase. Thus it becomes impractical to use in the real world scenario. A real world element is also brought into this heuristic - i.e. each rescuer is given a randomly assigned group of rescuees/co-ordinates on the map. As can be seen in the next chapter of results, the nature of solutions/tours that this algorithm produces directly varies with the randomness of the position of the rescuees on the map. This makes its solutions unreliable.

It serves as a good tool for comparison with the Disaster Recovery Heuristic, since we consider the same real world conditions with the thesis algorithm.

3.1.3 Disaster Recovery Heuristic

This algorithm first takes 'm' number of rescuees and 'n' number of rescuers. Each of the '$m-1, m$' rescuee have been assigned randomized distances between them. This is an alternative to assigning (x, y) positions to each rescuee. Now we assign each rescuer a set of rescuees, just as the previous algorithm. Each set of rescuees is called a group. Thus there will be 'n' groups. The number of rescuees may or may not be equal in each group depending upon if the numbers are exactly divisible or not. The we cycle through each rescuer creating a tour. Each tour is created by using a well established algorithm for creating tours called the GENI algorithm. This algorithm is described in great detail in chapter 2.

For the thesis algorithm, this is how the GENI algorithm is implemented -
For each rescuer, an initial tour is created. The initial tour in each case will be '$0, m-x, 0$' where '$m-x$' is the first rescuee to be assigned to that rescuer. '0' is the location of the depot. This signifies that each tour must begin and end at the depot. Then the more vertices/rescuees are added to the tour by using a particular insertion method prescribed by the GENI algorithm.

Thus 'n' tours are constructed. Now the total tour for the whole map is calculated. Although the GENI algorithm which is well proven[11] to provide good results, the tour can still be optimized. The optimization process involves taking into account all 'n' tours. Then the longest and the shortest tour are isolated. In the longest tour distance

between all cities are taken into account. The longest distances between two cities say '$x, x+1$' is identified. Similarly in the shortest tour, the smallest distance between two cities say '$y, y+1$' is identified. Then city x is taken from its tour and inserted into the shortest tour. Likewise, city y is taken from the shortest tour and inserted into the longest tour. The total tour is then recalculated and compared with the previous result. This process is repeated until we get the best possible result or we run out of the preset number of iterations, in the interest of having a reasonable time taken for calculation. This process has an averaging effect on the results produced. In the next chapter, it can be seen that the optimization process ensures that the results have a very predictable deviation from the results as shown by the greedy algorithm.

In this chapter we have explored the work that has been put in to explore the algorithm that is implemented for this thesis, as well as the work put in to produce a good and fair baseline for comparing the results produced by the Disaster Recovery Heuristic. The thought behind implementing other algorithms for comparison is that the results should be oblivious to the hardware used to produce results.

In the next chapter a detailed comparative study of the results from all three algorithms will be made.

Chapter 4

Study of results

In this chapter the results of three mapping algorithms are examined and compared to each other. The three algorithms are Greedy Algorithm, Local Search Heuristic and Disaster Recovery Heuristic. All three heuristics have been produced with the same compiler and have been run on a PC with the same hardware to generate timing results. Comparison with other studies is in terms of distance. The distance units used here are proportional to real distances but do not represent a specific unit. Here are the performance results of the heuristic in this work:

4.1 Results for 5 Rescuers - Time vs. No. of Rescuees

In figure 4.1 and table 4.1 we are comparing the computation time taken to calculate the tour for 5 Rescuers and an increasing number of rescuees - ranging from 10 to 4000. As can be seen from Table and Graph 6.1, all three algorithms show similar results in time for a small number of rescuers 550 - 1000. Beyond this number of rescuers we can see that the Local Search Heuristic performs the best of the three. Greedy Algorithm and DRH have similar computation times.

d

Table 4.1: 5 Rescuers: Time vs. No of Rescuees

No. of Rescuees	Greedy-Algo	DRH	LSH
10	9	31	14
50	43	624	34
100	142	2319	76
250	821	42207	319
500	3324	127541	1055
750	8311	417935	2216
1000	18057	893550	3767
1250	53861	1679431	5860
1500	61952	2793134	8287
1750	73035	4382424	17556
2000	84946	6457100	25756
2250	99042	9155204	19543
2500	116349	12438464	56764
2750	130231	16421925	61146
3000	149209	21524010	63643
3250	168811	27010295	73162
3500	190789	33633957	77710
3750	213507	41329788	82890
4000	238560	50154336	91125

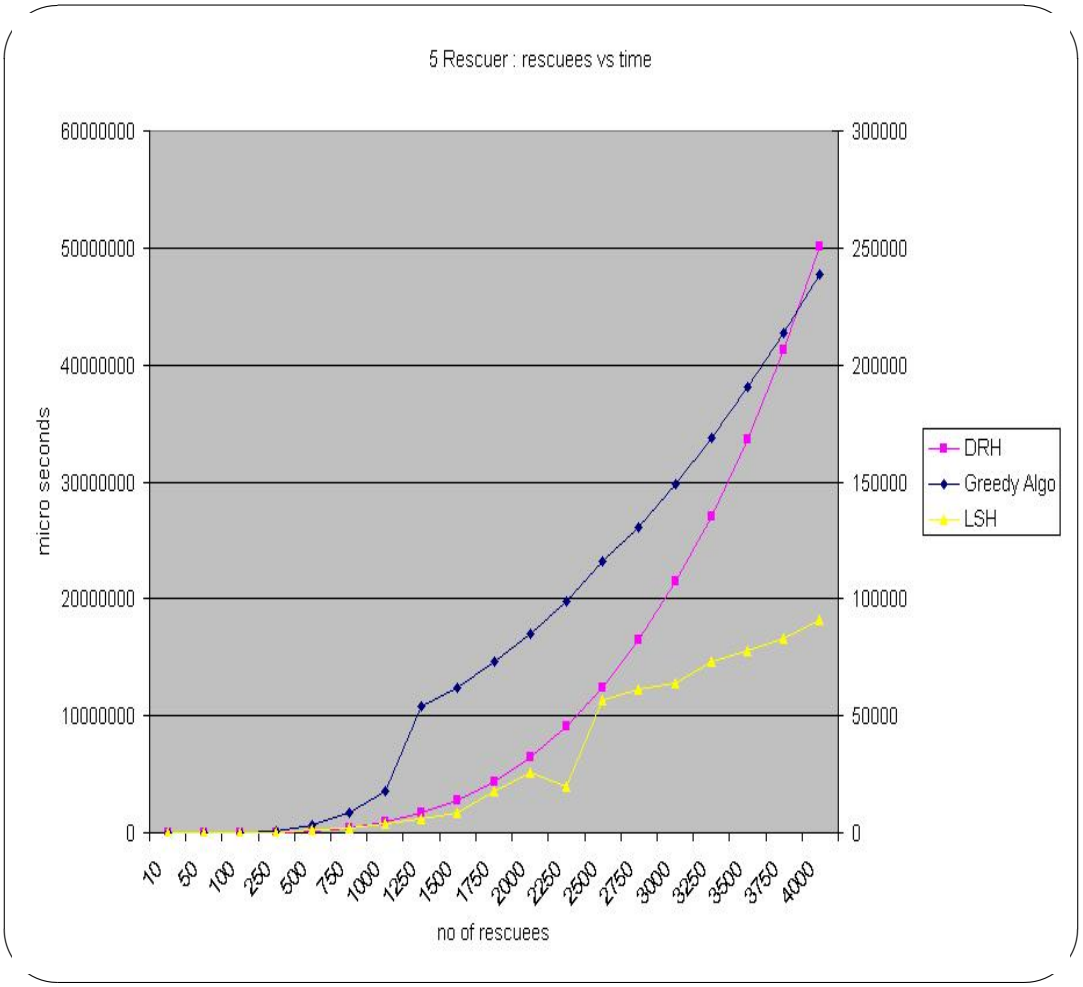

Figure 4.1: Graph: 5 Rescuers time vs. rescuees

4.2 Results for 5 Rescuers - Distance vs. No. of Rescuees

In figure 4.2 and table 4.2 we are comparing the results in terms of total distance of tour as calculated by the three algorithms. The Greedy algorithm for reasons explained earlier shows short distance results for the given map. We can see that Local Search Heuristic and DRH show similar deviation from the result as given by the greedy algorithm, for a large number of rescuees. However, the Local search heuristic which is subject to the randomness of local areas assigned to each rescuer and the randomness in the distances between rescuees/cities in each local area, shows less smoother results. At times because of this randomness we can even see that DRH is outperforming LSH for the total tour distance calculated for an increasing number of rescuees. Until we hit

3500 rescuees, the distance calculated by DRH shows a much smaller deviation than LSH, from the result generated by Greedy Algorithm.

Table 4.2: 5 Rescuers: Distance vs No.of Rescuees

No. of Rescuees	Greedy-Algo	DRH	LSH
10	6968	18917	13286
50	6552	62627	28006
100	11332	115575	36496
250	11335	271738	51625
500	13452	529459	47149
750	12186	834606	57022
1000	13007	1073761	54427
1250	15958	1320372	58779
1500	13603	1600701	59031
1750	17508	1865739	58625
2000	14907	2175299	64854
2250	15644	2415585	63122
2500	15537	2719853	68907
2750	16320	2969888	66156
3000	15444	3183072	71320
3250	17237	3461904	63926
3500	17082	3735212	64434
3750	18231	4037382	65843
4000	18378	4297058	68840

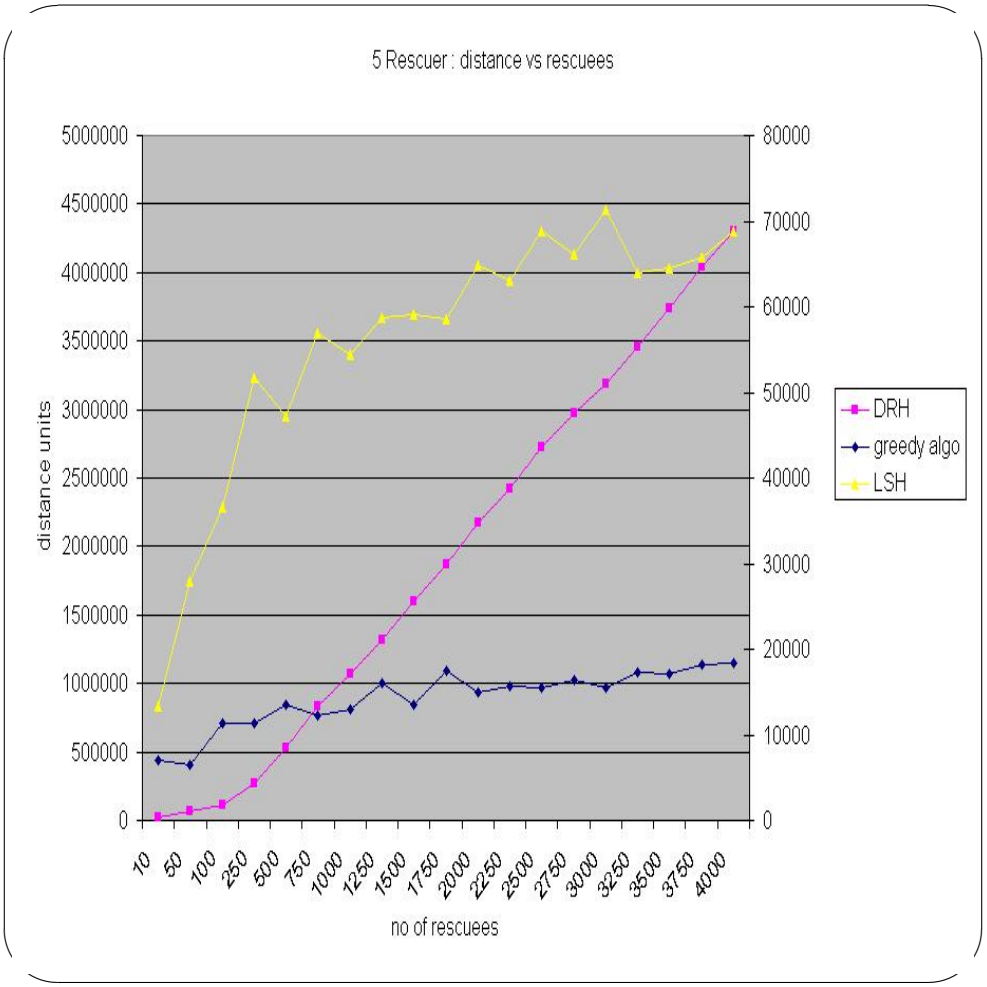

Figure 4.2: Graph: 5 Rescuers distance vs. rescuees

4.3 Results for 50 Rescuers - Time vs. No. of Rescuees

In figure 4.3 and table 4.3 we are comparing the computation time taken for 50 rescuers to calculate tours for 500 - 7500 rescuees. Once more it can be observed that all three algorithms have similar computation times for 500 to 3000 rescuees. Beyond this number, the Local Search Heuristic shows the best results of the three. DRH shows the largest computation time taken to calculate total tour.

Table 4.3: 50 Rescuers: Time vs. No. of Rescuees

No. of Rescuees	Greedy-Algo	DRH	LSH
500	3318	1562	265
1000	24288	5830	759
1500	62726	39762	1354
2000	84706	56772	2245
2500	101909	79512	3169
3000	151962	321414	4559
3500	191804	473140	6205
4000	239515	696070	7627
4500	296490	941168	9148
5000	357708	1250638	40441
5500	424473	1617578	46339
6000	495353	2053748	46715
6500	581469	2566961	49525
7000	668628	3171542	52996
7500	749831	3837999	55589
8000	844399	4568217	57222

4.4 Results for 50 Rescuers - Distance vs. No. of Rescuees

In figure 4.4 and table 4.4 we are comparing the total distance calculated for total tour. Results from DRH show the most deviation from the greedy algorithm result. LSH results come very close to the Greedy Algorithm results. This is a perfect case to show under which conditions DRH should NOT be used to generate a mapping solution. As seen in the graph both Greedy Algorithm and Local Search Heuristic give good results for relatively small numbers of rescuers and rescuees.

4.5 Results for 500 Rescuers - Time vs. No. of Rescuees

In figure 4.5 and table 4.5 we are comparing the computation time taken for 500 rescuers to calculate tours for 1000 to 9000 rescuees. The graph and tabular results show that the time taken for Greedy Algorithm increases exponentially for increasing number of rescuees. DRH and LSH show comparable results for time taken to compute total tours.

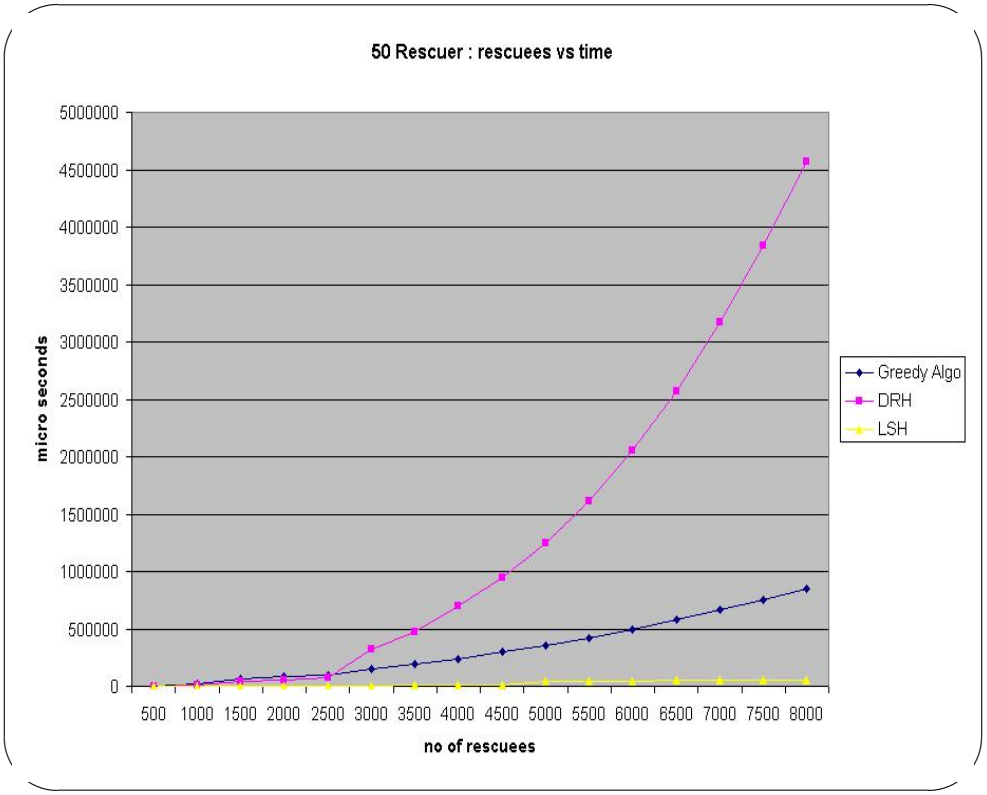

Figure 4.3: Graph: 50 Rescuers time vs. rescuees

Table 4.4: 50 Rescuers: Distance vs. No. of Rescuees

No. of Rescuees	Greedy Algo	DRH	LSH
500	13452	584332	594197
1000	13007	1135419	406607
1500	13603	1656490	404217
2000	14907	2241359	405125
2500	15537	2743744	541023
3000	15444	3258944	441444
3500	17082	3779552	456399
4000	18378	4278027	643853
4500	16355	4825386	484550
5000	18211	5422346	566846
5500	16998	5989713	524711
6000	17502	6591486	606152
6500	15911	7068793	655525
7000	15768	7558393	531817
7500	16092	8118259	556625
8000	17768	8624845	541496

Table 4.5: 500 Rescuers: Time vs. No.of Rescuees

No. of Rescuees	Greedy Algo	DRH	LSH
1000	17161	1733	720
1500	62078	2679	813
2000	84998	3750	975
2500	113502	4700	1193
3000	149711	6098	1430
3500	190673	8112	1713
4000	240832	32445	1995
4500	293926	20373	2497
5000	356304	39258	2671
5500	429735	42737	3009
6000	473158	46406	3614
6500	574852	51126	3842
7000	680779	30051	4531
7500	763303	55224	4953
8000	850309	58951	5325
8500	950083	67582	5844
9000	1055628	71434	6472

4.6 Results for 500 Rescuers - Distance vs. No. of Rescuees

In figure 4.6 and table 4.6) we are comparing the distances calculated for total tours by the three algorithms. Graph 6.6 shows an interesting variation in the results shown by LSH. Once again the graph line for LSH varies because of the randomness in the assigned local areas to rescuers and the randomness in the assigned distances to each rescuee/city. DRH on the other hand shows less deviation in its results from the Greedy Algorithm results upto 5000 rescuees and thereafter the results of LSH and DRH are somewhat comaparable if we take their mean deviations into account.

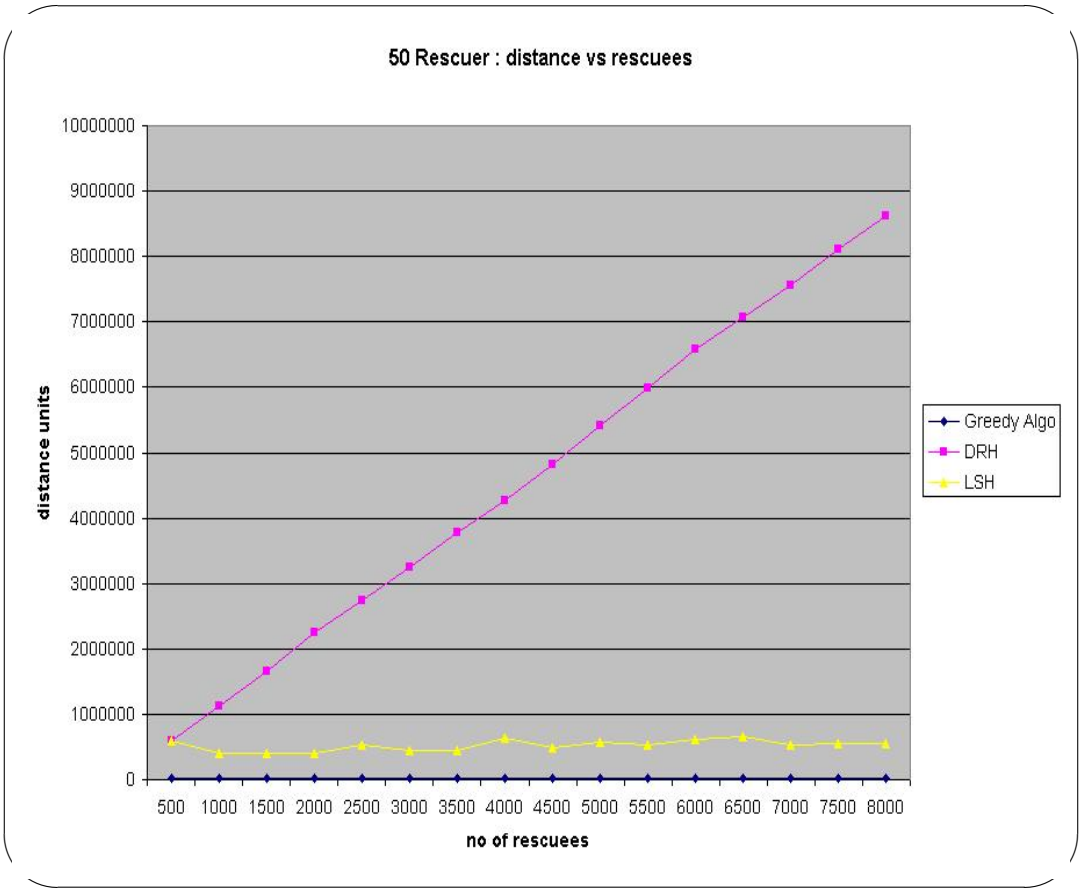

Figure 4.4: Graph: 50 Rescuers distance vs. rescuees

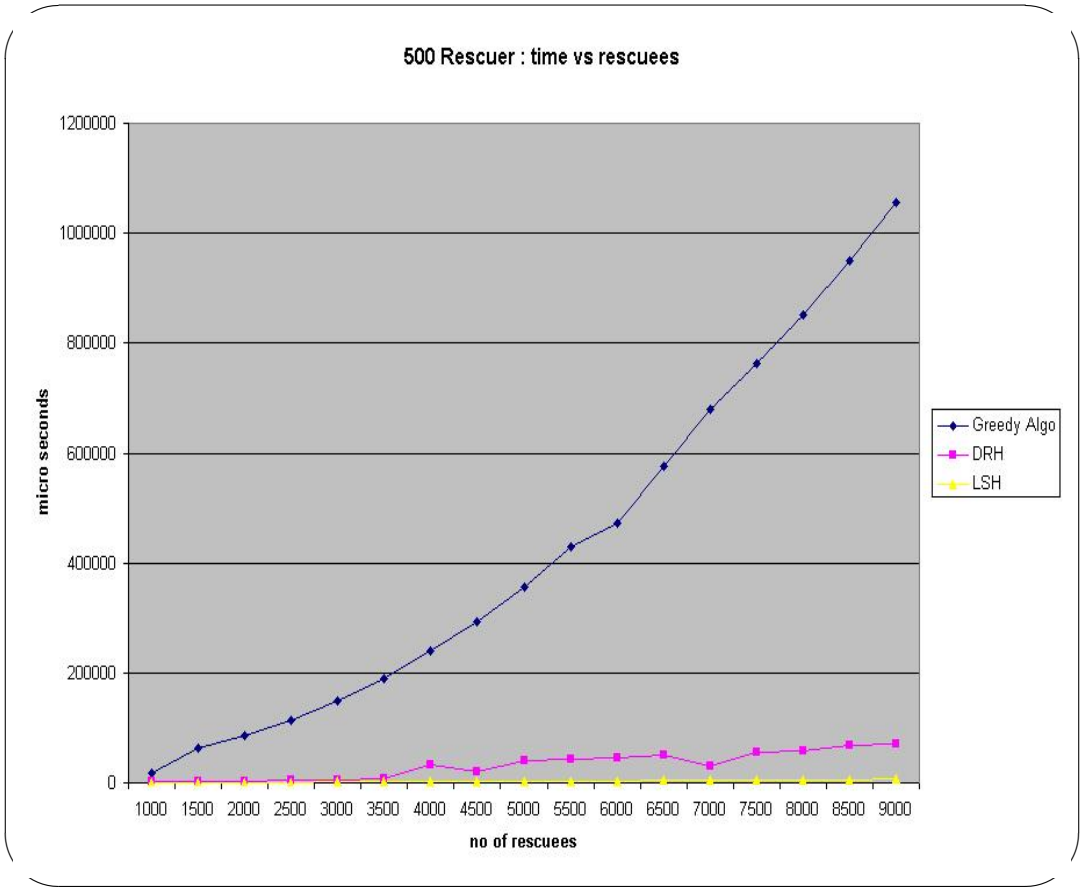

Figure 4.5: Graph: 500 Rescuers time vs. rescuees

4.7 Results for 1000 Rescuers - Time vs. No. of Rescuees

In figure 4.7 and table 4.7 we are comparing the time taken to compute the total tours for 1000 rescuers for 3000 - 12000 rescuees. As shown by Graph 6.7 the time taken for computation by the greedy algorithm increases exponentially. LSH and DRH show similar results for time taken to compute total tours.

Table 4.6: 500 Rescuers: Distance vs. No. of Rescuees

No. of Rescuees	Greedy Algo	DRH	LSH
1000	13007	1607452	4500691
1500	13603	2167154	4953070
2000	14907	2703842	15123647
2500	15537	3213360	7089110
3000	15444	3742969	5752508
3500	17082	4288470	4513975
4000	18378	4731703	9231707
4500	16355	5405293	5124510
5000	18211	5895861	6945157
5500	16998	6438612	4403998
6000	17502	7061688	4853179
6500	15911	7503136	3947188
7000	15768	7940430	4936134
7500	16092	8552182	3925351
8000	17768	9067873	5006670
8500	18005	9623988	4460826
9000	16672	10205802	5344224

4.8 Results for 1000 Rescuers - Distance vs. No. of Rescuees

In figure 4.8 and table 4.8 we are comparing the results of all three algorithms in terms of distances of total tours. LSH once again shows a very wide difference in the results for different number of rescuees. DRH on the other hand shows an increasing trend in the total distance calculated and shows a consistent deviation from the Greedy Algorithm results.

Table 4.7: 1000 Rescuers: Time vs. No. of Rescuees

No. of Rescuees	Greedy-Algo	DRH	LSH
3000	121919	5195	1540
3500	191781	6041	1693
4000	239326	7594	1975
4500	297707	8315	2319
5000	355441	9634	2616
5500	432627	31784	2902
6000	501002	33049	2957
6500	581476	36390	3178
7000	658261	37624	3597
7500	750561	41313	3909
8000	849144	42700	4247
8500	952699	46874	4584
9000	1070996	49141	4819
9500	1166039	53730	5268
10000	1290466	56069	5861
10500	1430999	62120	5996
11000	1543033	63093	6396
11500	1674138	68686	7220
12000	1822201	70202	8222

Table 4.8: 1000 Rescuers: Distance vs. Rescuees

No.of Rescuees	Greedy-Algo	DRH	LSH
3000	15444	4246307	8272232
3500	17082	4867556	10448271
4000	18378	5312644	26426766
4500	16355	5883622	16061640
5000	18211	6377715	13840207
5500	16998	6963648	9374478
6000	17502	7562742	15663569
6500	15911	8047784	10844919
7000	15768	8598036	10123438
7500	16092	9082365	14383307
8000	17768	9623598	16172968
8500	18005	10261919	11373382
9000	16672	10753266	9681225
9500	17005	11200349	9612204
10000	17256	11694088	11076374
10500	15760	12328997	8174724
11000	16874	12863273	8137332
11500	18164	13393698	7912334
12000	15937	13938564	8762235

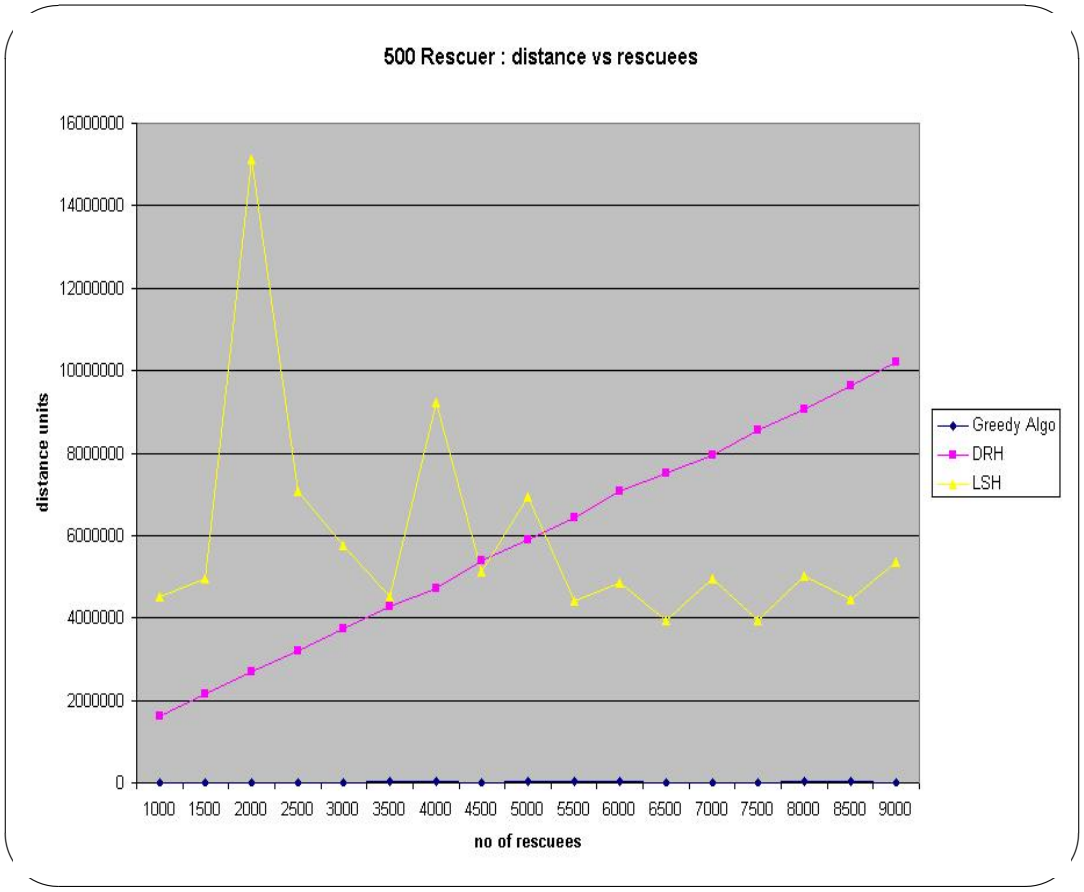

Figure 4.6: Graph: 500 Rescuers distance vs. rescuees

4.9 Results for 2000 Rescuers - Time vs. No. of Rescuees

In figure 4.9 and table 4.9 we are comparing time taken for computing tours for 2000 rescuers for 4000 - 16,000 rescuees. Here graph 6.9 shows that all three algorithms, have comparable results in terms of time upto 15,000 rescuees. Beyond this number Greedy Algorithm's computation time increases exponentially. LSH and DRH show similar results for time.

Table 4.9: 2000 Rescuers: Time vs. No. of Rescuees

No. of Rescuees	Greedy Algo	DRH	LSH
4000	209058	6963	2319
4500	310724	8069	2440
5000	363306	8756	2653
5500	422930	9291	3008
6000	499476	27285	3160
6500	544493	29472	3290
7000	667879	30419	3621
7500	750992	33440	3771
8000	850670	32401	4085
8500	953630	35409	4388
9000	1060441	36115	4652
9500	1185488	37480	4768
10000	1289136	38534	5342
10500	1401018	42443	5467
11000	1567081	43196	5815
11500	1680516	45637	6620
12000	1845910	46755	8266
12500	2087452	51103	8643
13000	2191261	51608	9349
13500	3666839	55757	34506
14000	3411442	80808	35510
14500	3190499	37449	36550
15000	86172719	2980844	2969513
15500	1.52E+08	7879556	7193803
16000	2.18E+08	9780370	10054384
16500	2.74E+08	13828625	13507119

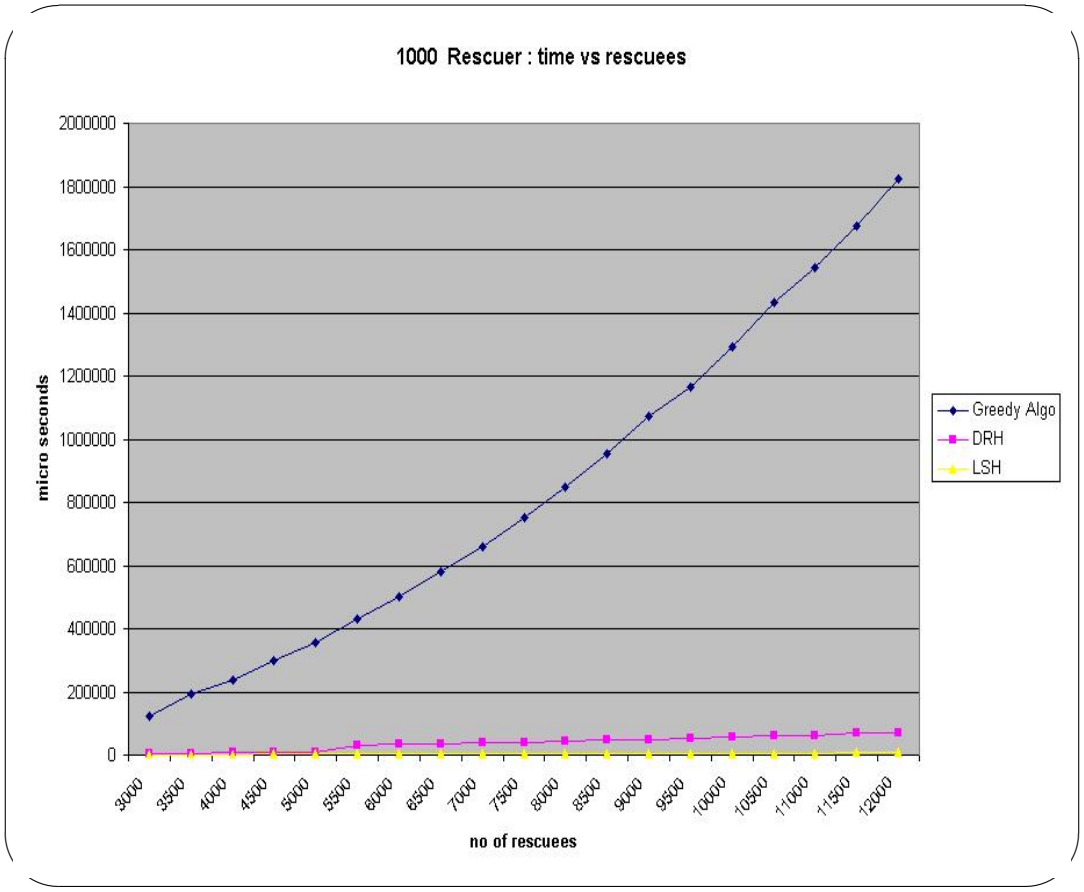

Figure 4.7: Graph: 1000 Rescuers time vs. rescuees

4.10 Results for 2000 Rescuers - Distance vs. No. of Rescuees

In figure 4.10 and table 4.10 we are comparing the total tour distances as calculated by all three Algorithms. Once again the results from LSH show a large variation in the results. DRH shows a consistent deviation from the results as shown by the Greedy algorithm. When compared to LSH results in graph 6.10, DRH results show great improvement in the quality of results.

Table 4.10: 2000 Rescuers: Distance vs. No. of Rescuees

No. of Rescuees	Greedy Algo	DRH	LSH
4000	18378	6426359	60920678
4500	16355	7003288	28720954
5000	18211	7484626	36791458
5500	16998	8052612	32620223
6000	17502	8669595	35942206
6500	15911	9079746	21846874
7000	15768	9569340	35567404
7500	16092	10226056	32467644
8000	17768	10670551	32135043
8500	18005	11198169	22559671
9000	16672	11770695	19850037
9500	17005	12200275	17721791
10000	17256	12769320	15794926
10500	15760	13427269	17841732
11000	16874	13913625	22690250
11500	18164	14479313	28500779
12000	15937	14994413	20111026
12500	16170	15506968	16133620
13000	16007	16140785	17007918
13500	17687	16638051	14778643
14000	16692	17224325	13069424
14500	15775	17738154	15917633
15000	15348	18226865	19810001
15500	16933	18721393	16017663
16000	18389	19264742	17138005
16500	15184	19859459	13935384

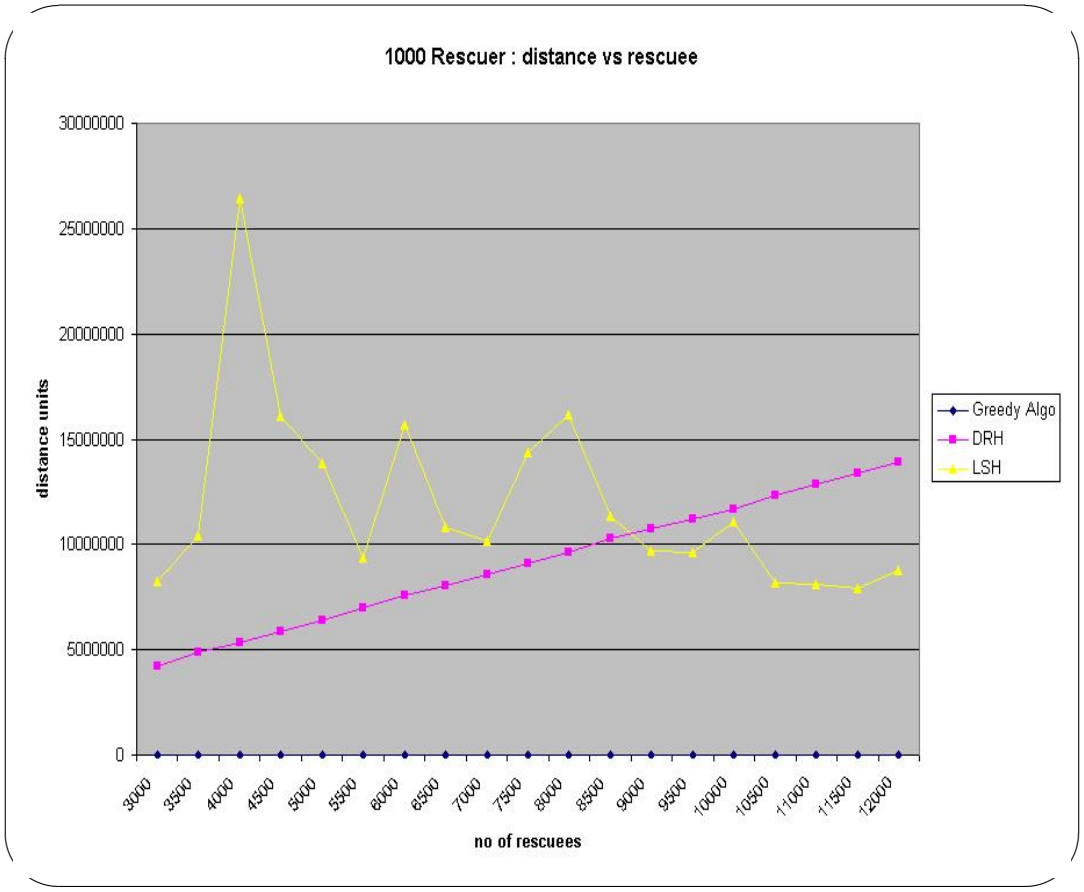

Figure 4.8: Graph: 1000 Rescuers distance vs. rescuees

4.11 Thesis results vs Outside study: Distance

In figure 4.11 and table 4.11 we can study the calculated distances produced by the algorithms implemented in the thesis and the algorithms introduced by the paper. Unfortunately, the paper only gives three data points i.e. 30 cities, 100 cities and 500 cities. From graph 4.11 we can see that the implemented Greedy algorithm has similar results to all of the algorithms in the study. This tells us that the implementation of the greedy algorithm has been correct and is a good baseline. The DRH computation results for distance, as studied in the previous sections shows poor results for a small number of cities. This is evident in the following table and graph.

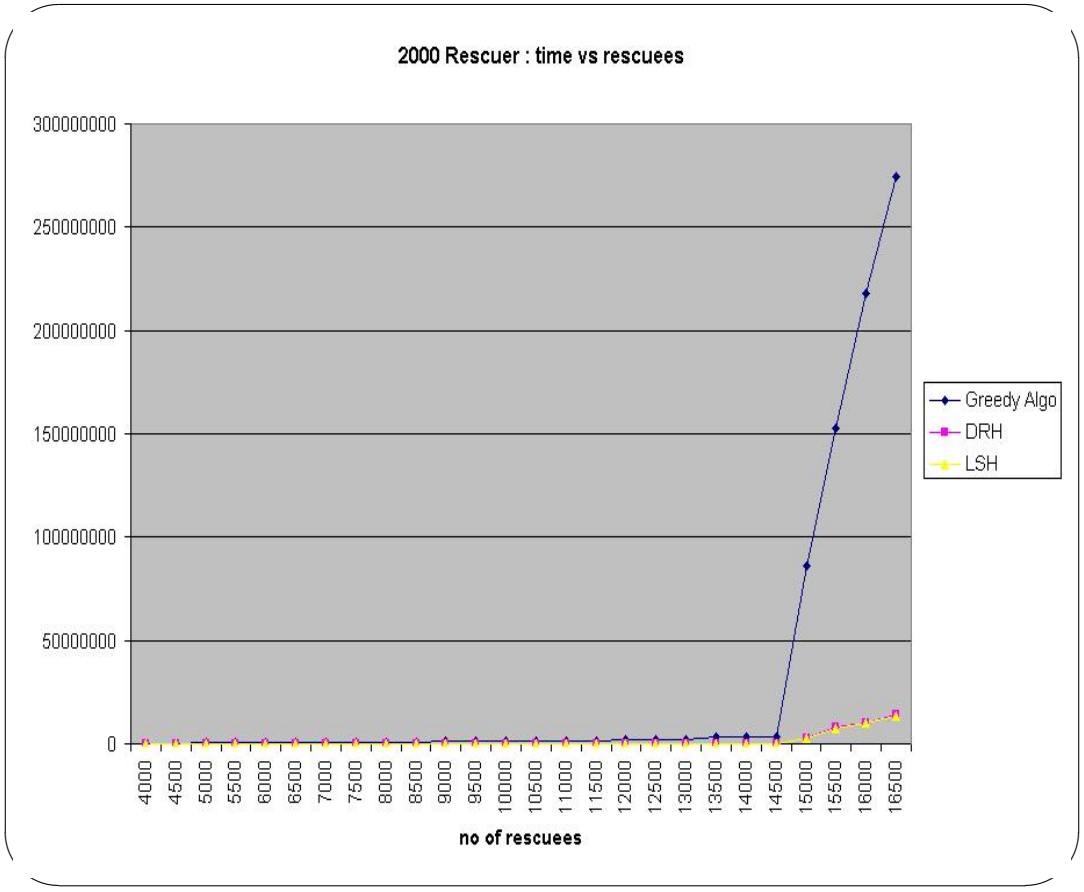

Figure 4.9: Graph: 2000 Rescuers time vs. rescuees

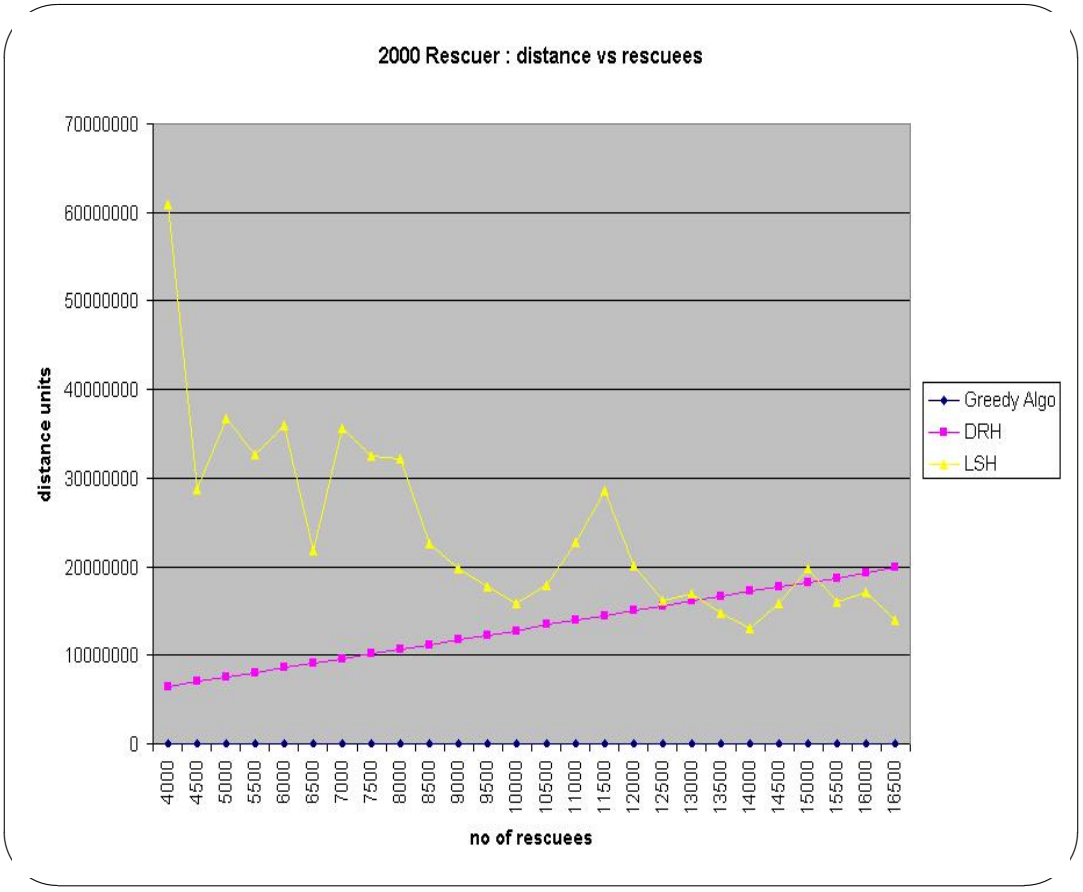

Figure 4.10: Graph: 2000 Rescuers distance vs. rescuees

Table 4.11: Thesis Results vs Other TSP Algorithms: Distance

Cities	Greedy Algorithm	DRH	LSH	Greedy	Greedy 2-Opt	2-Opt	Greedy 3-Opt	3-Opt	Simulated Annealing	Genetic Algorithm	Neural Network
10	6968	18917	13286								
30				5825	5237	5825	5600	5445	5442	5737	5215
50	6552	62627	28006								
100	11332	115575	36496	11195	10942	10800	10890	10890	10800	10755	10087
250	11335	271738	51625								
500	13452	529459	47149	26005	25317	25207			25697	24765	21945
750	12186	834606	57022								
1000	13007	1073761	54427								
1250	15958	1320372	58779								
1500	13603	1600701	59031								

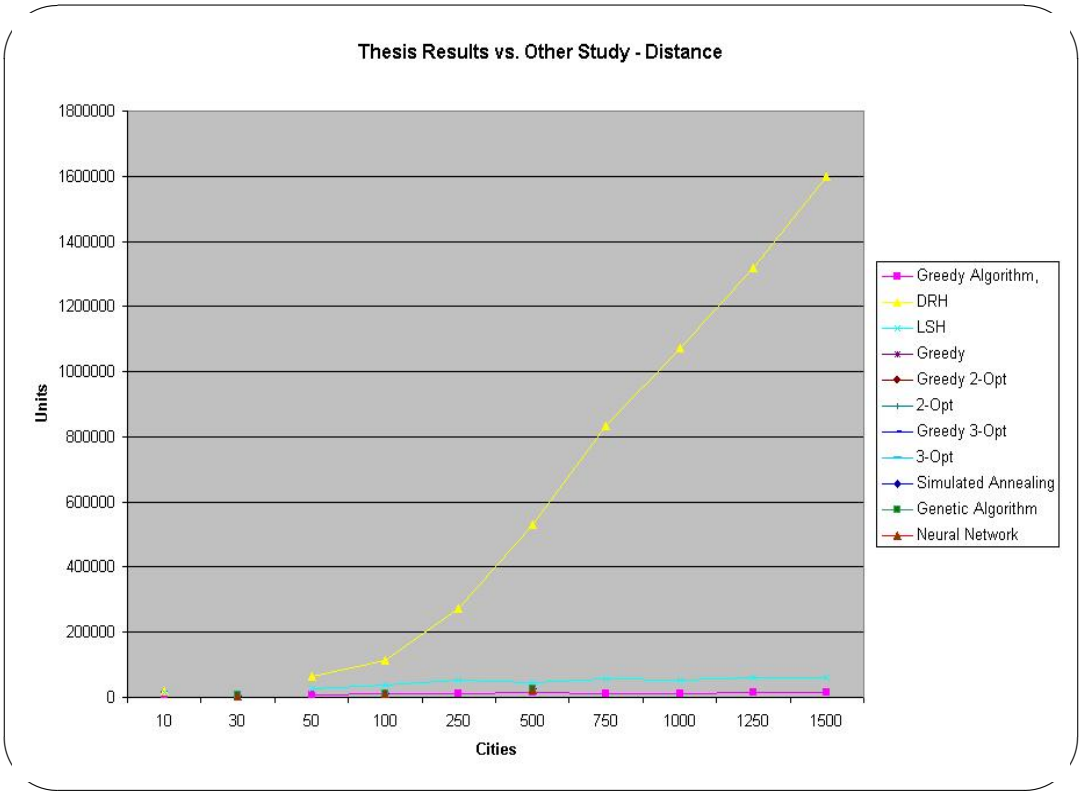

Figure 4.11: Graph: Thesis Results vs. Results from study 'Comparison of TSP Algorithms'

4.12 Summary of results

For distance vs no. of rescuees comparisons, it can be seen that the Greedy Algorithm produces comparitively good results. However as the number of rescuers and rescuees increase i.e. for number of rescuers greater than 500 and number of rescuees greater than 5000 4.5, the time for computing these results increases exponentially. Local Search Heuristic inherits the exponential search time for a large number of rescuers and rescuees. This algorithm also suffers from a major drawback, in that it depends on the variations of the map to produce results. Another look at graph 4.8 shows that the quality of mapping solutions varies directly with the randomness of the weights attached to the rescuees. In other words if some rescuees are farther away than some other rescuees on the map, the result will vary similarly. Thus this mapping solution does not produce predictable results. It has been implemented here as a mathematical model of an ad-hoc approach that most closely mimics a real-world scenario. On the other hand the thesis algorithm 'Disaster Recovery Heuristic' produces consistently predictable results in terms of distance of total tour and time taken for calculation. Graphs 4.9 and 4.10 both demonstate this. This predictability of results makes DRH very usable in the real-world scenario.

Chapter 5

Conclusion

This thesis was aimed at formulating a reasonably good method of creating a mapping solution, such that a given number of tours are mapped at a reasonably low cost (time and distance). It is aimed at being able to provide a tool to aid in a large rescue effort. Mapping such tours out before the recscue effort takes place allows for more predictable and planned results.

In finding a solution for this thesis, a lot of research in the field of the Travelling Salesman Problem was explored. Some of the ideas for solutions in this research was transposed to the implemented thesis problem. In the research portion, the papers that were explored were the Classic - TSP, Period TSP, Multiple salesmen-TSP, Local search heuristic and the Ant Colony TSP. Each of these provided unique points of view to the Disaster Recovery problem. The Classic TSP defines the problem of traversing a map in the least costly method in a very basic way, this creates a foundation for the thesis problem. From Period TSP it was observed that to create an effective solution temporal and distance planning is required. The Multiple salesmen - TSP showed how to co-ordinate multiple rescuers to create an overall optimal solution for the whole map. Local Search Heuristic showed how a map can be divided up into smaller regions, and also showed how achieving good results for each locality provides good results for the whole map (min-max objective). Finally from the Ant Colony - TSP, proposals for future expansion of this work can be made which will be explored in greater detail in the section called Proposal for future implementation.

The Disaster Recovery problem was then formulated to mimic a real life disaster scenario. There are multiple number of rescuers to traverse the map of a large number of rescuees. All rescuers have to start and end tour at the depot/city 0. Some rescuers have a longer route to traverse than the others. After formulating the the Disaster Recovery

Heuristic, a hybrid solution was created from all the solutions that were recommended by the research papers. This hybrid solution is called the Disaster Recovery Heuristic. Results generated by the Disaster Recovery Heuristic were then compared to the results generated by the Greedy algorithm (Classic - TSP) and Local Search Heuristic.

After a detailed study of results (in chapter 4) from the three algorithms (Greedy Algorithm, Local Search Heuristic (LSH) and Disaster Recovery Heuristic (DRH)) we can see that for a small number of rescuers and rescuees, Greedy algorithm and LSH show better results both in terms of time taken for computation and distance of total tours.

However for a larger number of rescuers and rescuees (i.e. for no. of rescuers greater than 500 and no. of rescuees greater than 1500), DRH shows comparitively good results. It shows a scalable deviation in terms of distance, from the greedy algorithm distance results. It also shows very comparable results to LSH in terms of time taken for computation. For larger numbers (greater than 5000) rescuees, there is good reason to believe that even LSH computation time will increase exponentially since it inherits the drawbacks of an exhaustive search algorithm. In the case of DRH, the computation time does not increase exponentially since it does not do an exhaustive search to construct its tours. Instead it relies on the well proven GENI method[11] to create its tours. The optimization following the individual tour creation, has an averaging effect on the solution. This allows it to be fairly reliable for large numbers of rescuers and rescuees.

From these results it is safe to conclude that with the Disaster Recovery Heuristic we can get reasonable results for constructing tours for a large number of rescuers and rescuees, in a comparably good period of calculation time.

THE MAJOR CONTRIBUTIONS OF THIS THESIS ARE:

1. Application of TSP to the defined real-world scenario.
2. Evaluation of other works in related fields.
3. Unique algorithm which produces optimum tours with the given parameters of the problem.

5.1 Proposal for future implementation

In this work we have proven that with the knowledge of distances between rescuees and number of rescuers an initial tour can be created. By executing each of these rescue

tours, an overall good rescue operation (in terms of speed of tour) can be mapped. However in the real-world scenario, situations are rarely if ever perfect and constant. Thus to create a solution that will truly work for a real-world scenario, we have to factor in changes that will inevitably occur during the execution of the initial tours. This can be done in the following ways:

1. In the GENIUS algorithm a tabu list was created. Once a tour is traversed, it is put into a tabu list. This is done so that this tour is not traversed again. In the Ant Colony - TSP it was explored how an arc was marked as desirable after it had been traversed. The more desirable the arc, the more probability it has to be chosen for a tour. Both of these methods provide an excellent way to reduce the number of iterations that are made to find the shortest tour, making a difference in the computation time required to come up with a good overall solution. This principal can be used for the Disaster Recovery Heuristic as well. In the thesis solution each rescuer was assigned a tour to be executed. During the tour execution, the quality of the tour can also be marked as desirable or not so desirable. According to this real-time data, optimizations can be carried out to create alternate situations.

2. Another adjustment that can be made is to use the optimization that has been implemented in DRH, in real-time. In the real-world scenario if one rescuer(team) has a shorter/easier tour than the other, then this can be averaged out by transferring some of the rescuees of the longest tour to the shortest tour.

Both of these optimizations in real-time, can contribute to make DRH better suited to adapt to a real-time environment. This increases the usability of the Disaster Recovery Heuristic.

Appendix

This part contains the flow graph of the implemented algorithm. It shows how initial tour is constructed and then optimized depending on time and resources available.

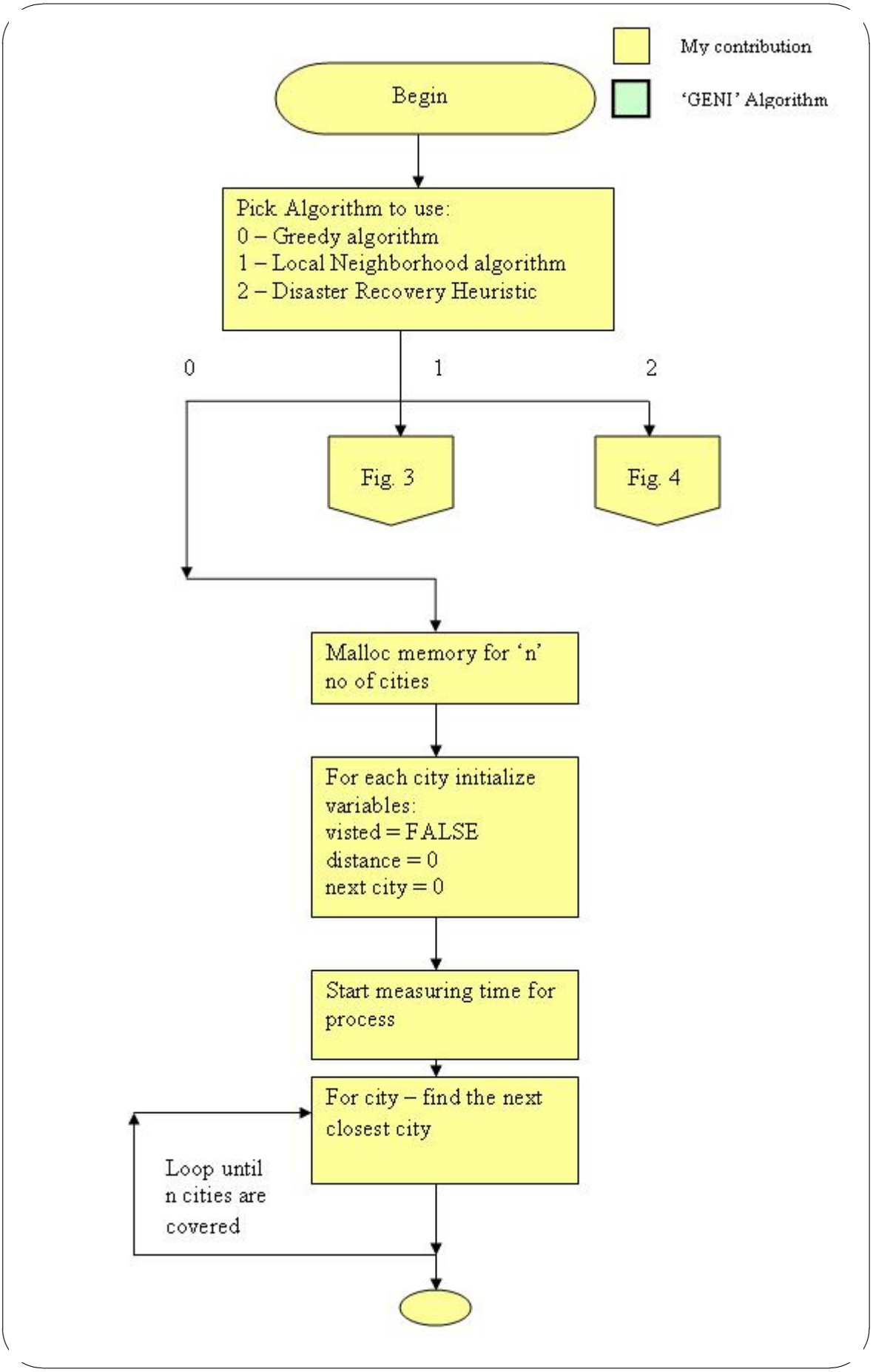

Figure 1: Diag 1 of 6

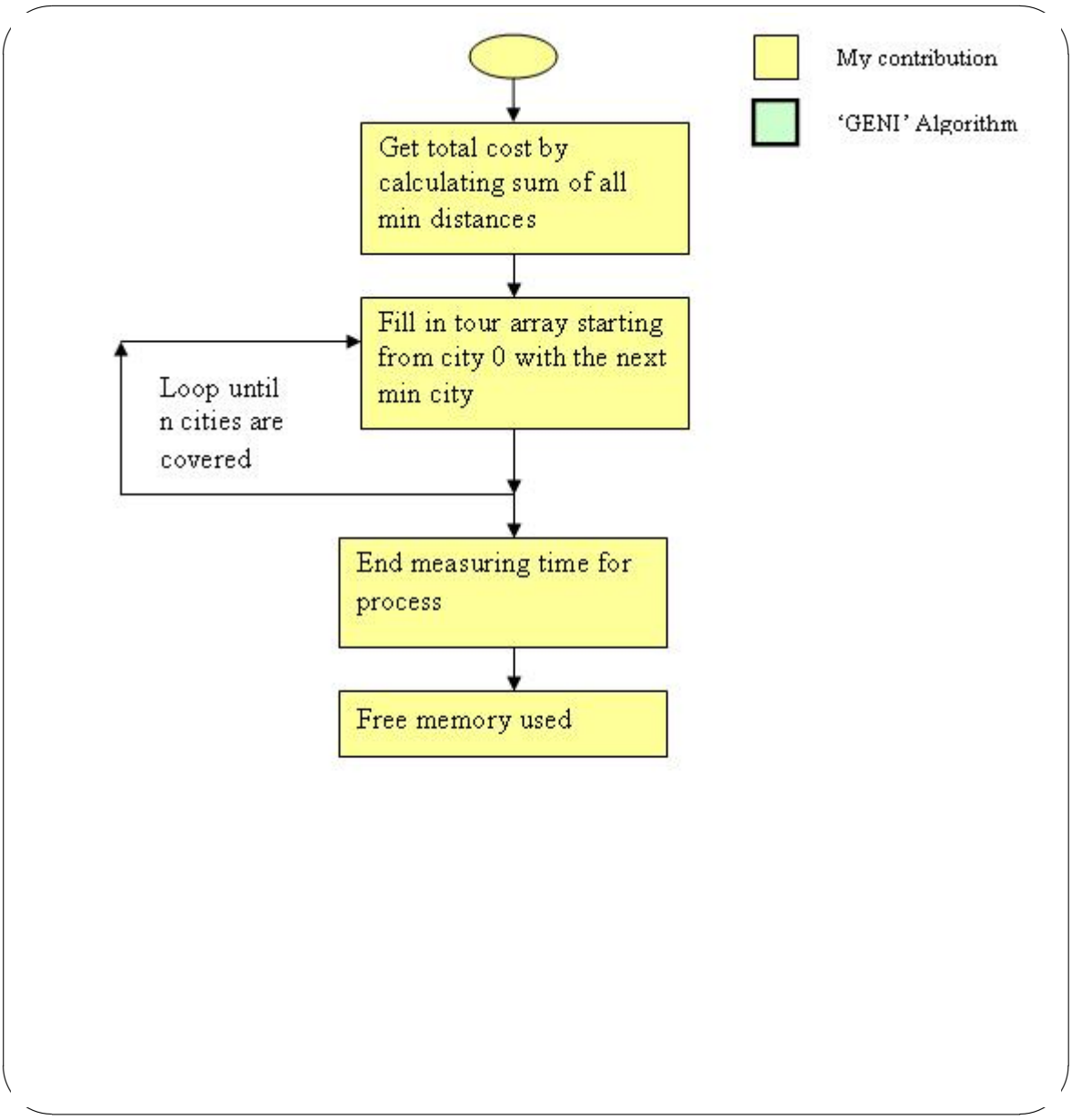

Figure 2: Diag 2 of 6

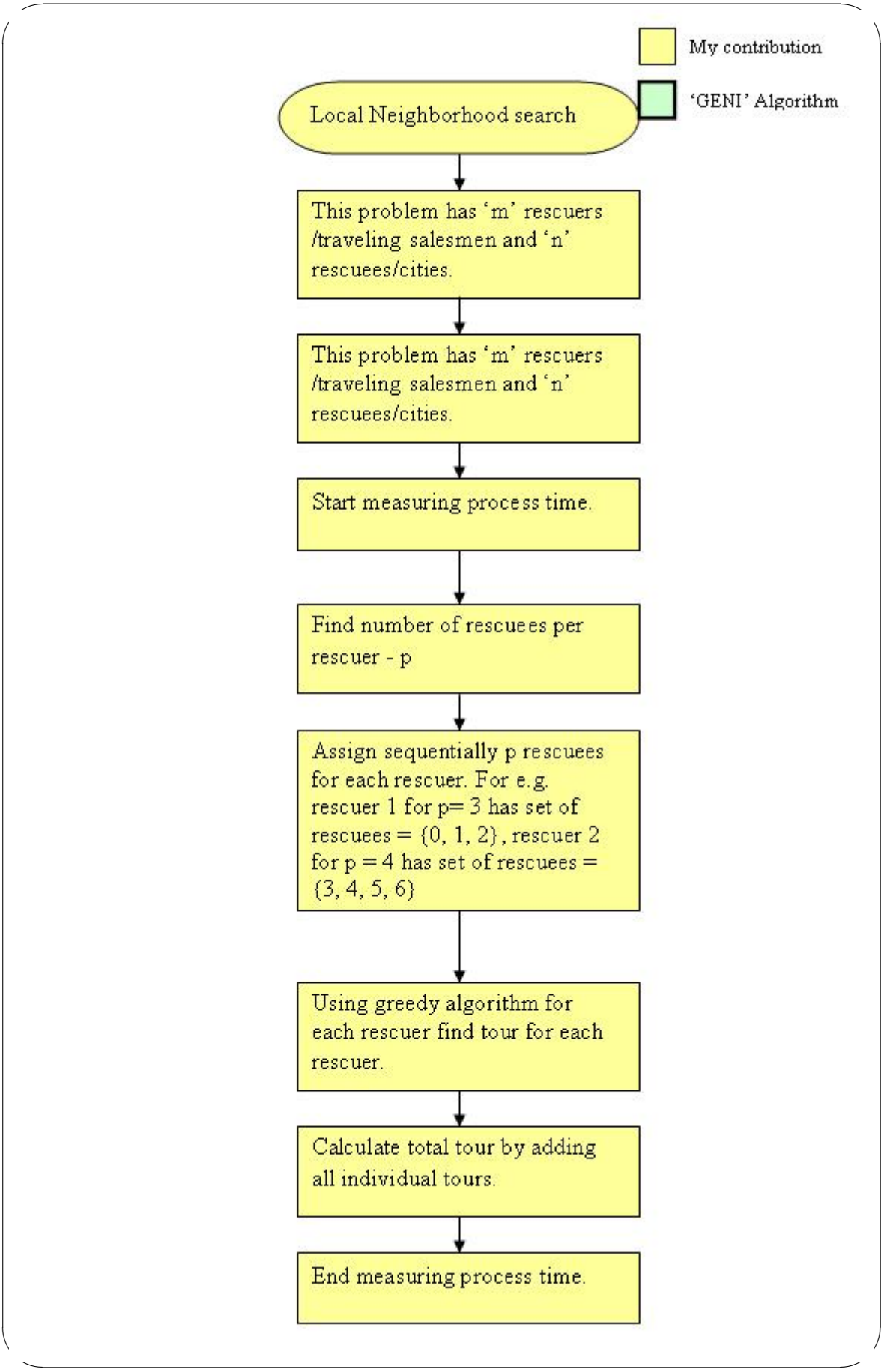

Figure 3: Diag 3 of 6

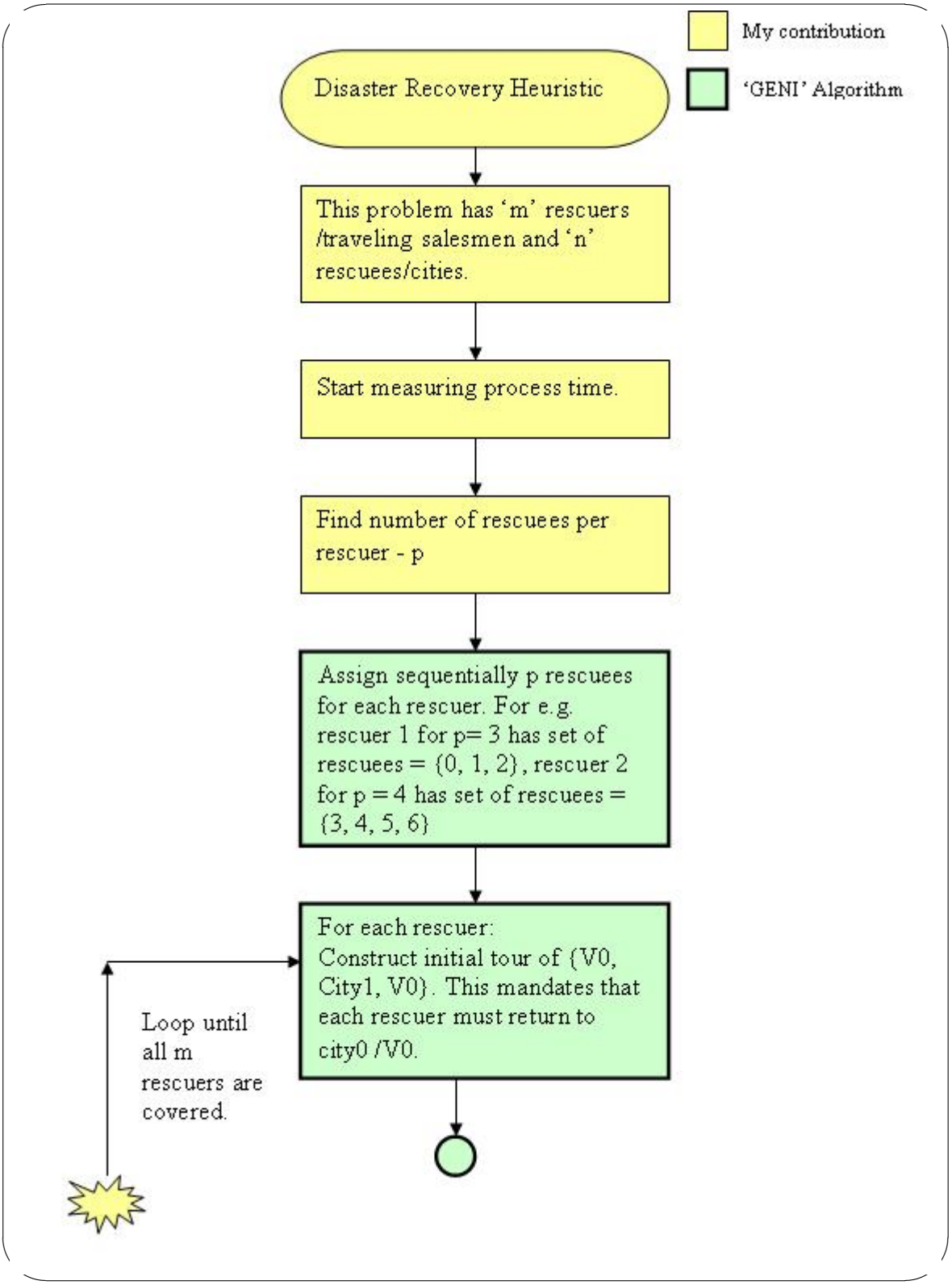

Figure 4: Diag 4 of 6

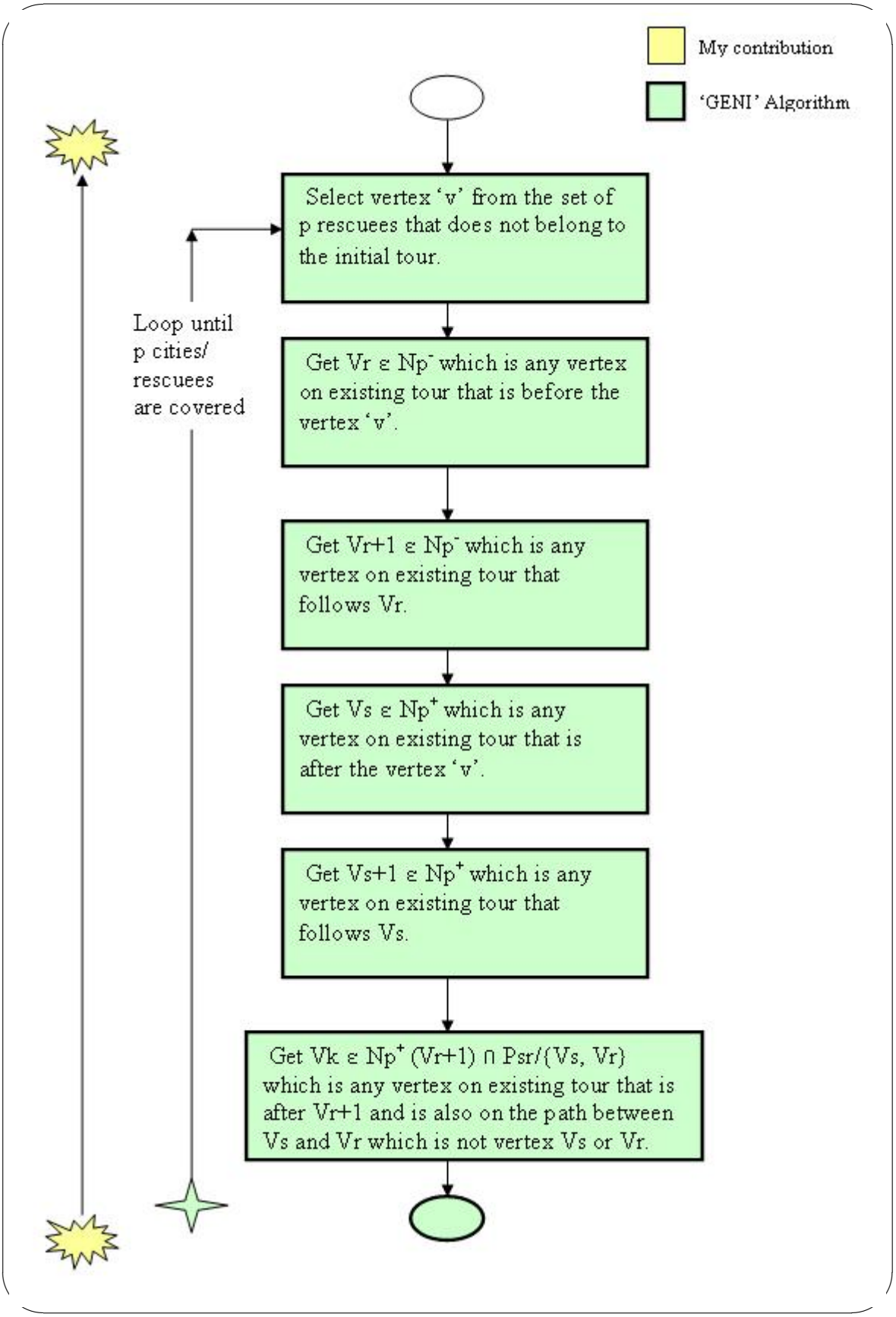

Figure 5: Diag 5 of 6

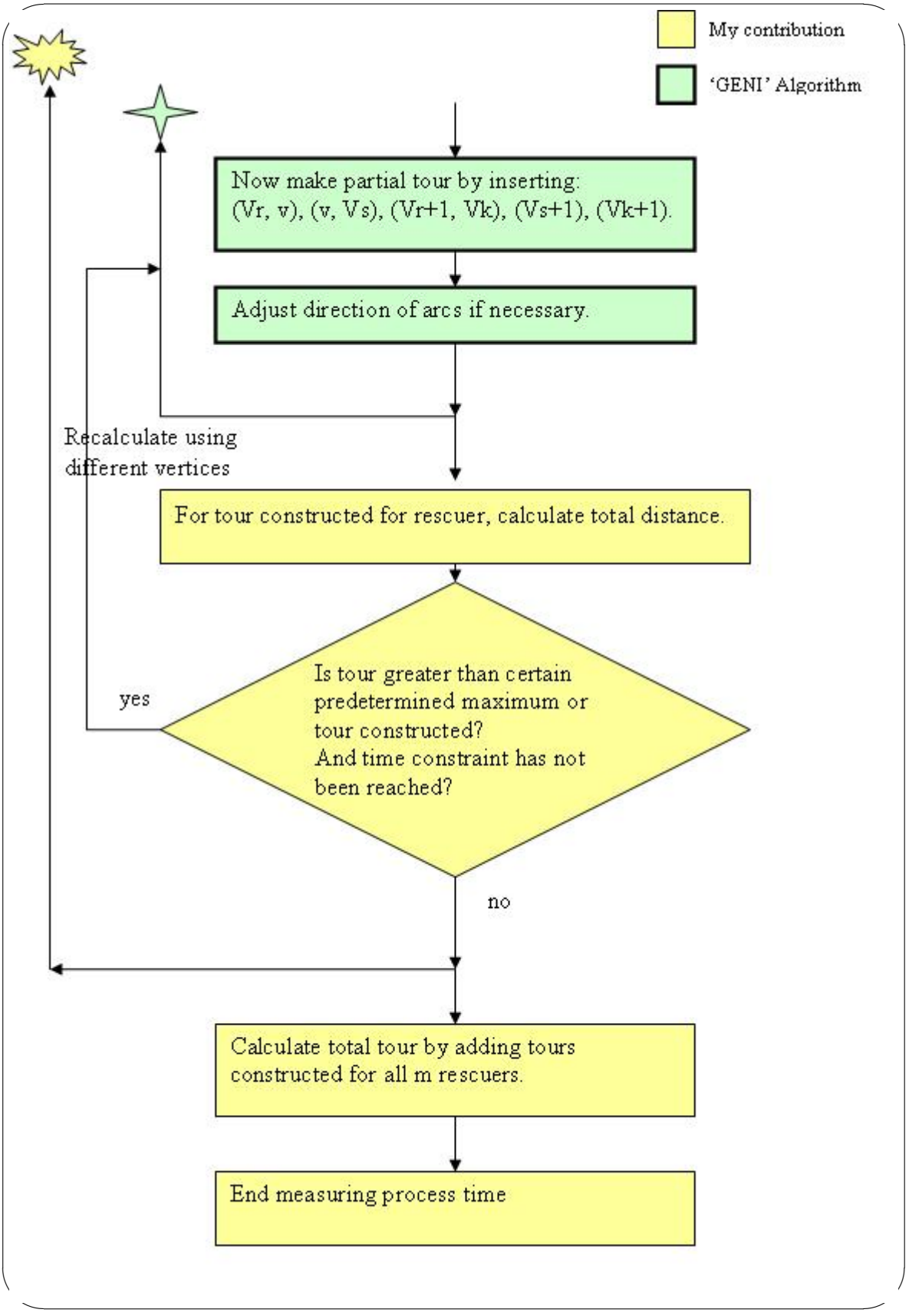

Figure 6: Diag 6 of 6

Bibliography

[1] Min Zhang Byung-In Kim, Jae-Ik Shim. Comparison of TSP Algorithms. Project for Models in Facilities Planning and Materials Handling, 1998.

[2] Mario Gerla Chunhung Richard Lin. Adaptive Clustering for Mobile Wireless Networks. IEEE Journal of Selected Areas in Communications, 1997.

[3] Robert Dakin. The travelling salesman problem. Technical report, PCUG, 1996-2002.

[4] Dantzing and Ramser. Vehicle routing problem - introduction. Technical report, NEO: Networking and Emerging Optimization, 1959.

[5] Giri Narasimhan Esther M. Arkin, Joseph S. B. Mitchell. Resource-Constrained Geometric Network Optimization. Symposium on Computaional Geometry, 1997.

[6] Atsushi Iwata Ching-Chuan Chiang Guangyu Pei Mario Gerla and Tsu wei Chen. Scalable routing strategies for ad hoc wireless networks. Technical report, CS Dept., UCLA, 1999.

[7] Gregory Gutin. Exponential neighbourhood local search for the travelling saleman problem. Technical report, Univ of West London, U.K., 1999.

[8] Giuseppe Paletta Luca Bertazzi and M. Grazia Speranza. An improved heuristic for the period traveling salesman problem. Technical report, Universita di Brescia, Italy and Universita della Calabria, Italy, 2004.

[9] Eric Taillard Luca Maria Gambardella and Giovanni Agazzi. Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. Technical report, IDSIA, Lugano, Switzerland, 1999.

[10] Giuseppe Paletta. The period tsp: A new heuristic algorithm. Technical report, Universita delgi Studi della Calabria, Italy, 2004.

[11] Gilbert Laporte Paulo M. Franca, Michel Gendreau and Felipe M. Muller. The m-travelling salesman problem with minmax objective. Technical report, Universidade Estadual de Campinas, Brazil, Universite de Montreal and Universidade Federal de Santa Maria, Brazil, 1995.